



School of  
Engineering

InIT Institut für angewandte  
Informationstechnologie

## **Bachelorarbeit Informatik**

### Neuartiges Orbitarium Darstellungskonzept

---

**Autoren**

Remo Preuss  
Nicolas Schaller

---

**Hauptbetreuung**

Prof. Dr. Karl Rege

---

**Datum**

18.06.2020

## **Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering**

### **Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering**

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

**Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.**

**Ort, Datum:**

Zürich, 18.06.2020

Bassersdorf, 18.06.2020

**Name Studierende:**

Nicolas Schaller

Remo Preuss

## Zusammenfassung

Die ZHAW hat ein System zur Darstellung der Oberfläche der Erde, ein sogenanntes «Orbitarium», vom Technorama Winterthur übernommen. Darunter ist ein Globus zu verstehen, auf welchem mittels (Innen-)Projektoren verschiedene globale Szenarien dargestellt werden können. Das vor 10 Jahren entwickelte System entspricht leider nicht mehr dem aktuellen Stand der Technik. Insbesondere die Erstellung der Inhalte ist umständlich und zeitaufwendig und erfolgt offline, d.h. der Inhalt muss mit Hilfe von herstellerspezifischen Werkzeugen vorgängig in ein proprietäres Format konvertiert werden. Unsere innerhalb der Abschlussarbeit im IT Studiengang entwickelte Lösung hingegen ermöglicht es, beliebige globale Darstellungen aus dem Internet, wie z.B. die des Wetters (Windy) oder Flugverkehrs (Flightradar24) direkt zu übernehmen und in Echtzeit auf dem Orbitarium zu visualisieren. Einerseits entfällt dadurch die zeitaufwendige Aufbearbeitung der Daten und Visualisierungen können von anderen Anbietern übers Internet d.h. verteilt zur Verfügung gestellt werden. Andererseits werden auch interaktive Szenarien ermöglicht: der Benutzer kann direkt auf die Darstellung Einfluss nehmen, womit sich ein ganz neues Anwendungsgebiet eröffnet. Dieses wird in einer Nachfolgearbeit (Corona Visualisierung) genauer untersucht.

## **Abstract**

ZHAW School of Engineering obtained a System from the Science Centre “Technorama” in Winterthur, a so called “Orbitarium” for depicting the surface of our planet Earth. This is effectively a globe on which's inside global sceneries are being projected through the means of multiple projectors. Unfortunately, this 10-year-old system doesn't conform to the state of the art anymore. Especially the content creation step is cumbersome, time consuming and offline, which means that the content has to be converted beforehand and with the use of manufacturer-specific tools into a proprietary format. The solution of this final assignment of the IT major provides the possibility to use any applicable source from the Internet e.g. global wheater (Windy) or air traffic (FlightRadar24). It is visualized dynamically and in real time onto the “Orbitarium”. On the one hand this makes the expensive task of pre-authoring the data redundant and enables the consumption of third-party visualisations from the internet. On the other hand this opens up the possibility for interactive visualisations: the user may interact directly with the content projected onto the “Orbitarium” which in turn opens up a whole new area of application. This will be examined further in a follow-up project (Corona Visualisation).

# Vorwort

Die Erde ist unser aller Heimat und der Planet, welchem alles uns bis anhin bekannte Leben entspringt. Sie bietet uns ein Zuhause und ist ein Garant für spektakuläre Ereignisse. Ob Tsunamis, Vulkanausbrüche, die Eiszeiten oder die durch das Virus SARS-CoV-2 ausgelöste weltweite Pandemie: Die Erde bildet eine Theaterbühne, welche einen Schauplatz für allerhand mögliche oder auch vermeintlich unmögliche Ereignisse bietet.

Die Darstellung dieser Naturphänomene hat uns schon immer fasziniert. Die zweidimensionale Darstellung mittels Karten ist ein erster Schritt, doch die Darstellung auf einem dreidimensionalen Objekt ermöglicht ganz neue Möglichkeiten. Natureignisse lassen sich in ihrer richtigen Grösse und ihrem gesamten Ausmass darstellen. Dieser Umstand hat uns enorm motiviert und uns während der gesamten Arbeit begleitet. Wir hoffen, den Grundstein für viele zukünftige Animationen auf dem Orbitarium gelegt zu haben und wünschen allen Betrachtern viel Spass beim Betrachten und Erfahren des blauen Planeten (oder auch anderer runder Himmelskörper).

Diese Arbeit wäre ohne Unterstützung nicht möglich gewesen. Wir möchten uns ganz herzlich bei folgenden Personen und Institutionen bedanken:

- Bei **Herrn Prof. Dr. Karl Rege**: Für den Anstoss der Beschaffung des Orbitariums durch die ZHAW Winterthur vom Technorama Winterthur, für die Hilfe bei der Lösung von Problemen jedweder Art (vor allem bei den unvorhersehbaren), für die Unterstützung der Durchführung der Bachelor-Arbeit unter erschwerten Bedingungen und zu guter Letzt für stets erheiternde Video-Konferenzen im Home-Office.
- Beim **Technorama Winterthur**, namentlich **Herrn Jörg Moor**: Für den stetigen und unkomplizierten Zugang zum Technorama und somit zum Orbitarium, für den Wiederaufbau des eigentlich bereits demonstrieren Orbitariums, für die Flexibilität hinsichtlich des Zeitpunktes der Abholung des Orbitariums und die stets angenehme Zusammenarbeit.
- Bei **Herrn Thomas Hofer**: Für die stets zeitnahen, ausführlichen Rückmeldungen bei Fragen zur Technik des Orbitariums.
- Bei der Firma **Lang Baranday**, namentlich **Herrn Adrian Kälin**: Für die Beratung hinsichtlich geeigneter Projektoren.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>8</b>
1.1 Ausgangslage . . . . .	8
1.2 Stand der Technik . . . . .	8
1.2.1 ARC Science: OmniGlobe . . . . .	8
1.2.2 National Oceanic and Atmospheric Administration: Science On a Sphere . . . . .	9
1.3 Zielsetzung . . . . .	9
1.4 Anforderungen und Übersicht . . . . .	10
<b>2 Theoretische Grundlagen</b>	<b>11</b>
2.1 Das Orbitarium . . . . .	11
2.1.1 Das Projektorgehäuse . . . . .	12
2.1.2 Die Plexiglasröhre . . . . .	12
2.1.3 Die Projektionskugel . . . . .	12
2.2 Unity . . . . .	12
2.2.1 Shader in Unity . . . . .	12
2.3 Projektionen der Erde . . . . .	12
2.3.1 Rektangularprojektion . . . . .	13
2.3.2 Merkatorprojektion . . . . .	13
2.3.3 Azimutalprojektion . . . . .	14
2.4 Projektoren . . . . .	14
2.4.1 Anzeigeverfahren . . . . .	14
2.4.2 Weiterführende technische Aspekte . . . . .	16
<b>3 Vorgehen / Methoden</b>	<b>18</b>
3.1 Analyse und Evaluation . . . . .	18
3.1.1 Die bestehende Lösung . . . . .	18
3.1.2 Problemstellung und Risiken . . . . .	21
3.1.3 Wahl der Methode . . . . .	22
3.1.4 Evaluation der Tools . . . . .	23
3.2 Vision der Lösung . . . . .	24
3.3 Hardware . . . . .	25
3.3.1 Projektoren . . . . .	25
3.3.2 PC . . . . .	27
3.4 Software . . . . .	28
3.4.1 Die Darstellung der Animationen . . . . .	28
3.4.2 Die Quelle des Inhalts für ManyCam . . . . .	29
3.4.3 ManyCam und Stream in Unity . . . . .	31
3.4.4 Shaderlösung mit Unity . . . . .	31
3.4.5 Ausgabe auf mehreren Bildschirmen . . . . .	32
3.4.6 Das Orbitarium Control Panel . . . . .	32
3.4.7 Implementierung des Orbitarium Control Panel . . . . .	34
<b>4 Resultate</b>	<b>38</b>
4.1 Visualisierung mittels Orbitarium . . . . .	38
4.1.1 Versuchsaufbau . . . . .	38
4.1.2 Versuchsvorbereitungen . . . . .	39

4.1.3	Versuchsdurchführung . . . . .	39
4.1.4	Versuchsresultate . . . . .	39
<b>5</b>	<b>Diskussion / Ausblick</b>	<b>40</b>
5.1	Interpretation und Mehrwert . . . . .	40
5.2	Resultat im Vergleich zur Aufgabenstellung . . . . .	41
5.3	Ausblick . . . . .	42
<b>6</b>	<b>Verzeichnisse</b>	<b>44</b>
6.1	Literaturverzeichnis . . . . .	44
6.2	Abbildungsverzeichnis . . . . .	47
<b>A</b>	<b>Anhang</b>	<b>48</b>
A.1	Aufgabenstellung . . . . .	48
A.2	Code Listings . . . . .	49
A.2.1	Application.cs . . . . .	49
A.2.2	BrowserManager.cs . . . . .	51
A.2.3	ProjectionManager.cs . . . . .	53
A.2.4	SettingsManager.cs . . . . .	58
A.2.5	LogManager.cs . . . . .	61
A.2.6	ProjectionShader.shader . . . . .	61
A.2.7	maphelper.js . . . . .	63
A.2.8	openskydata.js . . . . .	63

# 1 Einleitung

Die Zürcher Hochschule für angewandten Wissenschaften hat vom Technorama Winterthur [1] ein sogenanntes «Orbitarium» beschafft. Darunter ist ein Globus zu verstehen, welcher durch den Einsatz von Projektoren in der Lage ist, verschiedene Szenarien auf der Erdoberfläche darzustellen. Das Orbitarium ist rund 10 Jahre alt und dementsprechend technisch veraltet. In dieser Arbeit soll das Orbitarium modernisiert werden.

## 1.1 Ausgangslage

Orbitarien sind ein Nischenprodukt und relativ teuer in der Anschaffung, wie am Beispiel des in dieser Arbeit behandelten Orbitariums unter Kapitel 3.1.1 gezeigt wird.

## 1.2 Stand der Technik

Um den aktuellen Stand der Technik zu eruieren, werden Anbieter von Orbitarien und deren Lösungen betrachtet. Da die hier vorgestellten Anbieter keine öffentlichen Dokumentationen zu ihrer Lösung zur Verfügung stellen, können wir nur soweit auf die jeweiligen technischen Details eingehen, wie wir sie aus den entsprechenden Unterlagen ersehen können.

### 1.2.1 ARC Science: OmniGlobe

ARC Science Simulations [2] vertreibt den sogenannten «OmniGlobe». Verschiedene Ansichten und Animationen der Erde können damit dargestellt werden. Der Globus ist auf einem Podest aufgestellt, in welchem 2 Projektoren montiert sind. Sie projizieren ihr Bild auf einen in der oberen Hälfte der Kugel platzierten Spiegel [3]. Dieses Bild wird dann zurück auf die Fläche des Globus projiziert. Der beschriebene Sachverhalt ist in Abbildung 1 dargestellt.

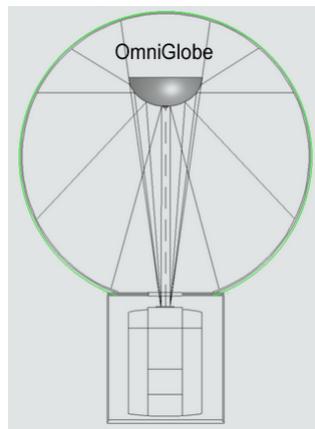


Abbildung 1: Projektionsgeometrie ARC Science OmniGlobe [3]

Mit dem OmniGlobe-System kommen 2 vorinstallierte Anwendungen [4]. Die Software «ARC Explorer» erlaubt es, eine Animation auszuwählen und neue Animationen zu erstellen. Die Software «ARC's Geometer» konvertiert existierendes Bild- oder Filmmaterial auf das spezifische, vom OmniGlobe benötigte Ausgabeformat.

### 1.2.2 National Oceanic and Atmospheric Administration: Science On a Sphere

Die NOAA ist die Wetter- und Ozeanografiebehörde der Vereinigten Staaten. Sie vertreibt das System «Science On a Sphere» [5], welches Animationen der Erde darstellen kann. Der Globus wird von 4 verschiedenen Projektoren von aussen angestrahlt [6]. Das mitgelieferte Softwarepaket umfasst viele verschiedene Anwendungen. Zu nennen ist die Software «SOS Display Software» für die Darstellung der Animationen, «SOS Visual Editor» für die Erstellung neuer Animationen sowie «SOS Public Kiosk» für die Bedienung des gesamten Systems [7]. Abbildung 2 zeigt einen beispielhaften Aufbau des Systems. Gut zu sehen ist einer der externen Projektoren auf der rechten Seite. Die NOAA verfügt ausserdem über eine Vielzahl an Animationen [8].



Abbildung 2: Eine «Science On a Sphere»-Installation [9].

### 1.3 Zielsetzung

Ausgehend von den bereits bekannten Umsetzungen (Kapitel 1.2) und der Aufgabenstellung (Siehe Anhang A.1, Seite 48) wurde das Ziel definiert, die Hardware zu erneuern und eine zeitgemässse Software zu entwickeln. Hierbei wurden folgende Kriterien für die Hard- und Software identifiziert:

- Hardware
  - Die Anforderungen für geeignete Projektoren sollen evaluiert werden. Die Projektoren sollen zeitgemäss Charakteristiken aufweisen und nach Möglichkeit direkt beschafft werden.
  - Um die Projektoren mit Bildmaterial zu beliefern, wird ein geeigneter Computer benötigt. Dieser soll evaluiert werden.
  - Das Orbitarium wird an der ZHAW aufgehängt. Die Aufhängepunkte müssen errechnet und eine geeignete Aufhängung muss evaluiert werden.

- Software
  - Die Software für den Globus soll erneuert werden.
  - Es sollen Darstellungen in Echtzeit berechnet werden können.
  - Es soll einen *Single Point of Entry* für das Gesamtsystem geben, von wo aus alles gestartet und gesteuert werden kann.
  - Das langwierige und aufwändige sogenannte «Pre-Authoring», also das Erstellen neuer Animationen, ist bei der aktuellen Version des Globus einer der Hauptkritikpunkte am gesamten System. Die Erstellung von Animationen soll in der überarbeiteten Version möglichst einfach sein.

Zusammenfassend soll in dieser Arbeit das System komplett erneuert werden. Dazu gehört die Evaluation und Beschaffung der Hardware sowie die Implementierung einer passenden Software.

## 1.4 Anforderungen und Übersicht

Um diese Arbeit verstehen zu können, wird vom Leser ein Grundverständnis der Informatik verlangt. Außerdem ist die Fähigkeit Grafiken, technische Zeichnungen und Tabellen zu interpretieren nötig. Des Weiteren sollte ein Verständnis für mathematische Formeln vorhanden sein. Mathematische Kenntnisse sind zwar nicht nötig, doch Grundlagen in Geometrie und Trigonometrie sind von Vorteil.

Die Arbeit gliedert sich in fünf unterschiedliche Teilbereiche. Die Kapitel sind der Reihe nach durchzulesen.

**Kapitel 2** beschreibt die wichtigsten theoretischen Grundlagen, auf welchen diese Arbeit aufbaut. Diese sind unter anderem das *Orbitarium* selbst oder die verwendeten *Projektionsarten* der Erde. Es ist auf den Seiten 11 - 18 zu finden.

**Kapitel 3** befasst sich mit den Methoden und dem Vorgehen zur Erreichung der Ziele. Der erste Teil befasst sich mit der Evaluation der eingesetzten Tools und des generellen Vorgehens inklusive Risikoabschätzung. Danach stellen wir eine Vision unseres Systems vor und erarbeiten in den 2 folgenden Unterkapiteln zuerst die nötigen hardwareseitigen und danach die softwareseitigen Lösungen. Dieses Kapitel ist auf den Seiten 18 - 38 zu finden.

**Kapitel 4** beschreibt die effektiven Resultate dieser Arbeit. Die mit der Softwarelösung berechneten Visualisierungen werden auf dem Orbitarium gezeigt. Zu finden ist dieses Kapitel auf den Seiten 38 - 40.

In **Kapitel 5** diskutieren wir die zuvor gezeigten Resultate kritisch. Danach geben wir einen Ausblick auf mögliche Erweiterungen, Verbesserungen und Features. Das Kapitel befindet sich auf den Seiten 40 - 44.

Im **Anhang** sind die Aufgabenstellung auf Seite 48 sowie die Quellcode-Listings ab Seite 49 zu finden.

## 2 Theoretische Grundlagen

Diese Arbeit basiert grösstenteils auf Verformungen und Projektionen von Bildern. Deshalb ist Wissen im Bereich «Visual Computing» notwendig. Um eine geeignete Entwicklungs-Engine zur Realisierung der neuen Orbitarium-Software sowie passende Hardware zur Berechnung und Projektion der Bilder evaluieren zu können, müssen die theoretischen Grundlagen zu unserem Orbitarium sowie die infrage kommenden Input- und Output-Projektionen analysiert und verstanden werden. Dieses Kapitel widmet sich diesen theoretischen Grundlagen.

### 2.1 Das Orbitarium

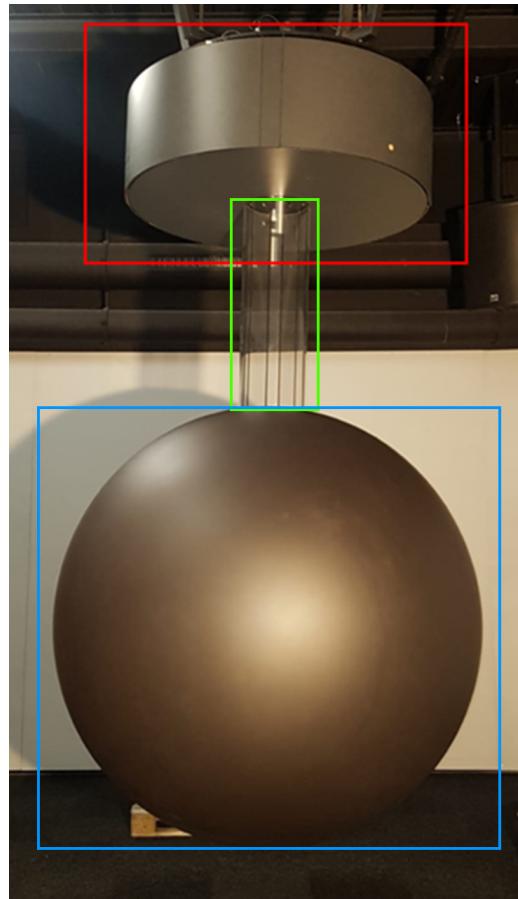


Abbildung 3: Orbitarium des Technorama. Ursprünglicher Platz und Aufhängung.

Das Orbitarium ist eine an der Decke aufgehängte Konstruktion. Es ist in Abbildung 3 dargestellt. Es besteht im wesentlichen aus den 3 folgenden Teilen:

- Das Projektorgehäuse (zuoberst, rot umrandet)
- Die Plexiglasröhre (mittig, grün umrandet)
- Die Projektionskugel (unten, blau umrandet)

### **2.1.1 Das Projektorgehäuse**

Das Projektorgehäuse enthält hauptsächlich die Beamer, welche «gegeneinander» projizieren und so ihr Bild auf die Projektionskugel werfen. Die genaue Projektionsgeometrie, also die Art und Weise, wie das Bild der Projektoren auf die Projektionskugel gelenkt wird, ist Teil der Analyse und wird im Kapitel 3.1.1 erklärt.

### **2.1.2 Die Plexiglasröhre**

Die Plexiglasröhre verbindet das Projektorgehäuse mit der Projektionskugel. In dunklen Umgebungen ist sie praktisch nicht sichtbar, weswegen beim Betrachter das Bild eines in der Luft schwebenden Globus entsteht. Um das Gewicht der Projektionskugel tragen zu können, ist die Plexiglasröhre mit Stangen aus schwarz lackiertem Stahl verstärkt.

### **2.1.3 Die Projektionskugel**

Die Projektionskugel ist eine Kugel aus mattem, schwarzem Plexiglas. Im Boden sitzt ein Spiegel, welcher das Licht aus der Plexiglasröhre reflektiert und auf das Projektionskugel umlenkt. Die Kugel ist schwarz lackiert, um beim Betrachten nicht zu viel direktes Licht vom Spiegel durchscheinen zu lassen.

## **2.2 Unity**

Unity ist eine plattformunabhängige Spieleentwicklungs-Engine. Obwohl Unity der Spieleentwicklung entstammt, bewährt es sich auf mehr als 25 Plattformen sowie auch im Bereich Film, Architektur, Konstruktion und in anderen Ingenieurwissenschaften. [10] Unity bietet ebenfalls Lösungen im Bereich *Virtual Reality* und *Augmented Reality*. Unity erlaubt einen leichten Einstieg, da skriptgesteuertes Verhalten in C# programmiert werden kann.

### **2.2.1 Shader in Unity**

Shader sind ein wesentlicher Bestandteil dieser Arbeit. Shader sind im Prinzip C#-Skripte, welche die mathematische Anweisung enthalten, wie jeder Pixel des Inputs im Output dargestellt wird. Dies bedeutet, dass der Input nicht direkt verändert wird, sondern viel mehr ein Shader *darüberlegt* wird. Somit geht keine Information bei der Transformation des ursprünglichen Bildes verloren. Im Kontext dieser Arbeit wird ein Shader verwendet, um das ursprüngliche Bild in eine Azimutalprojektion (erklärt in Kapitel 2.3.3) umzuwandeln.

## **2.3 Projektionen der Erde**

Da unsere Erde bekanntlich rund bzw. ein *Rotationsellipsoid* [11] ist, gestaltet sich die Darstellung in zweidimensionaler Form schwierig. Es gibt keine eindeutig beste Variante, um die Erde darzustellen. Auf die eine oder andere Weise muss immer ein Kompromiss gemacht werden. Flächen-, Winkel- sowie die Längentreue können in einer zweidimensionalen Darstellung der Erde nicht gleichzeitig erreicht werden. Dementsprechend sind die so genannten Projektionsarten vielfältig. Je nach Anwendung eignet sich eine andere Projektionsart besser. Einen Überblick der existierenden Projektionsarten liefert Wikipedia. [12] Nachstehend sind die drei Projektionsarten erläutert, welche für diese Arbeit von Bedeutung sind.

### 2.3.1 Rektangularprojektion

Die Rektangularprojektion, zu deutsch auch als Plaktkarte bekannt, ist eine längentreue Projektion. [13] Heutzutage erfreut sich diese Art der Projektion grosser Beliebtheit, da man mittels digitalen Systemen die geographischen Koordinaten direkt auslesen bzw. eintragen kann. Aus diesem Grund wird die Plaktkarte auch als die *Geographische Projektion* bezeichnet. Wie in Abbildung 4 zu sehen ist, sind die Quadrate des Gitternetzwurfs dieser Projektion alle exakt gleich gross. Die Längenkreise der Erde werden hierbei als vertikale Linien gezeichnet und die Breitenkreise dementsprechend als horizontale Linien. Die Eigenschaft der Längentreue bewirkt, dass der Abstand zwischen zwei *Längenkreisen* auf dieser Projektion identisch ist mit der realen Distanz, obwohl die Landmassen verzerrt und vergrössert dargestellt sind (Grönland wirkt grösser als Australien, in der Realität ist Grönland etwa 3 mal kleiner als Australien). Ausserdem gilt die Rektangularprojektion dank der besonders einfachen Beziehung zwischen den Pixeln im Bild und dem effektiven geographischen Ort als Basisreferenz für alle Umrechnungen in andere Projektionen.

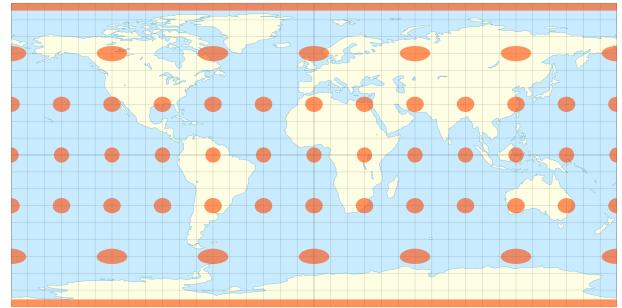


Abbildung 4: Rektangularprojektion mit tissotscher Indikatrix. Durch die Längentreue werden die orangenen Kreise an den Polen über das ganze Bild gleichmässig gezerrt.[13]

Ähnlich wie die *Rektangularprojektion* (beschrieben in Kapitel 2.3.1) ist die Merkatorprojektion eine Darstellung der Erde im rechteckigen Format. Anders als bei der *Rektangularprojektion* hingegen ist die Merkatorprojektion nicht längentreu, dafür aber winkeltreu.[13] Durch das Strecken der Erdkugel auf eine zweidimensionale Ebene wird die Nord-/Südachse ebenso ausgedehnt. Durch die Erhaltung der Winkeltreue dehnt sich diese so stark aus, dass die Pole unendlich gross sein müssten. Dieser Effekt ist in Abbildung 5 ersichtlich. Aus diesem Grund wird in der Regel alles über dem 70. Breitengrad (Nord und Süd) weggelassen.

### 2.3.2 Merkatorprojektion

Trotz der enormen Verzerrung der Landmassen basiert jede Seekarte auf der Merkatorprojektion, da sogenannte *Loxodrome* (Kurven auf einer Kugeloberfläche) als Gerade auf der Karte dargestellt werden können.

Die Merkatorprojektion bzw. eine Variante der Merkatorprojektion hat sich ebenfalls im Internet als Standard für Mapping-Tools etabliert. Dies ist hauptsächlich der Einführung von Google Maps 2005 [14] und ihrem Gebrauch der *Web Merkator*-Projektion zu verdanken.

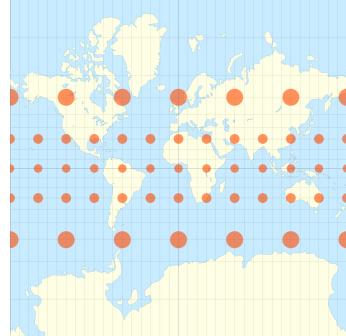


Abbildung 5: Merkatorprojektion mit tissotscher Indikatrix. Der Nord-/Südpol kann nicht dargestellt werden, da die Verzerrung an diesen Punkten unendlich gross ist.[13]

### 2.3.3 Azimutalprojektion

Die Azimutalprojektion wurde von dem Schweizer Physiker Johann Lambert entwickelt und gehört zu den flächentreuen Projektionsarten. Dies bedeutet, dass die Flächen der Formen, in diesem Fall der Kontinente, trotz starker Verzerrungen korrekt beibehalten werden. Die Lambertsche Azimutalprojektion findet ihre Anwendung bei der Umwandlung einer Kugeloberfläche zu einer zweidimensionalen Kreisfläche. Das Zentrum der Kreisfläche bleibt dabei unverzerrt, wohingegen der Bereich am Rand bis zur Unkenntlichkeit verzerrt wird (Abbildung 6).

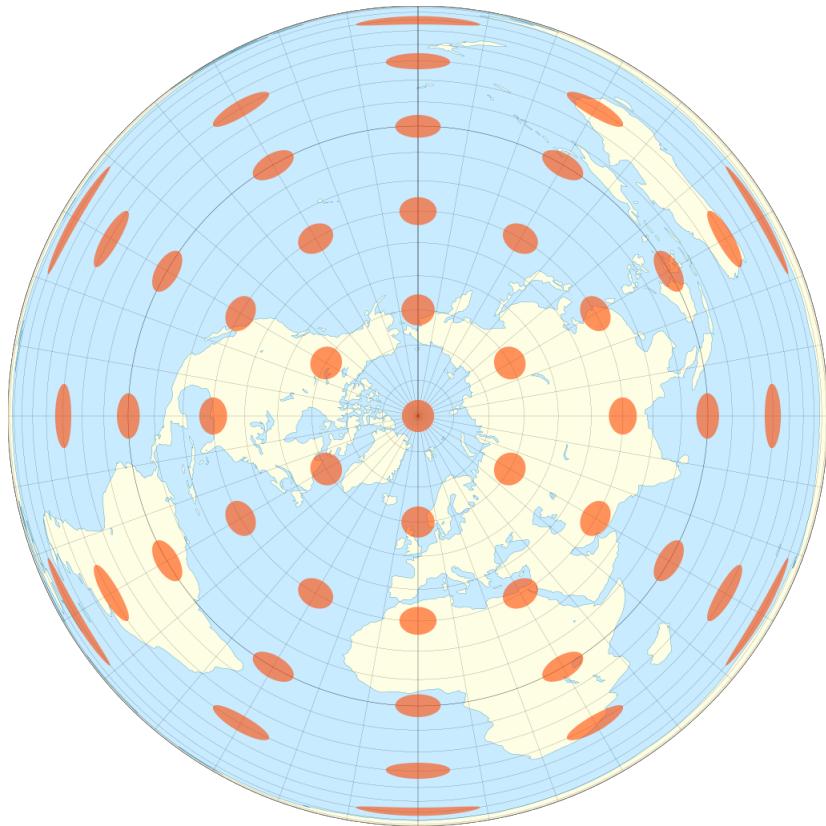


Abbildung 6: Azimutalprojektion der Erde mit eingezeichneter Tissotscher Indikatrix. Das unverzerrte Zentrum stellt der Nordpol dar. An der Kannte des Kreises kann man ganz dünn den stark verzerrten Südpol erkennen. [13]

## 2.4 Projektoren

Verschiedene Anwendungen benötigen verschiedene Projektoren. Es gibt auf dem Markt eine grosse Breite von Projektoren. Sie unterscheiden sich in verschiedenen Punkten voneinander. In diesem Kapitel soll ein Grundverständnis für den aktuellen Stand der Technologie sowie deren Charakteristika geschaffen werden.

### 2.4.1 Anzeigeverfahren

Unter den Anzeigeverfahren versteht man das Verfahren, wie das Bild erzeugt wird. Folgende Anzeigeverfahren sind zu nennen:

## LCD

Projektoren mit Flüssigkristallen funktionieren ähnlich wie Dia-Projektoren: Vereinfacht ausgedrückt durchfliesst Licht LCD-Panels und wird anschliessend durch die Linse an eine Leinwand projiziert. Pro Grundfarbe (Rot, Grün, Blau) gibt es ein LCD-Panel. Das Licht, das aus der Lampe austritt, wird durch sogenannte *diochroitische Spiegel* in die entsprechenden Bestandteile für das jeweilige LCD-Panel zerlegt. Das LCD-Panel lässt bei jedem Pixel eine bestimmte Menge Licht durch, genauso so viel, wie das Bild an diesem Pixel vom jeweiligen Farbbestandteil benötigt. Schlussendlich treffen die 3 Farbstrahlen in einem Prisma wieder aufeinander und stellen nun das Bild dar. [15] LCD-Projektoren haben den Vorteil, dass sie verschiedene Helligkeitsstufen sehr fein regeln können und kein Regenbogeneffekt auftritt. Als Nachteil ist zu nennen, dass LCD-Projektoren kein richtiges Schwarz projizieren können, da sich die LCD-Panels nicht komplett verdunkeln lassen. Zudem müssen die LCD-Panels immer in einem konstanten Temperaturfenster gehalten werden, damit der Inhalt der Panels im flüssigkristallinen Zustand bleibt und nicht in den flüssigen Zustand übergeht. Dieser Umstand macht leistungsfähige Kühlösungen nötig, die praktisch immer hörbar sind. Zu guter Letzt ist die Lampe nach etwa 4000 Stunden am Ende ihrer Lebensdauer angelangt.

## DLP

DLP heisst «Digital Light Processing». Die Farbe entsteht dadurch, dass das Licht von der Lampe durch ein Farbrad geleitet wird. Das Farbrad ist ein Rad mit mindestens den Grundfarben (Rot, Grün und Blau). Neuere Geräte besitzen mehr Farben. Es gilt: Je mehr Farben auf dem Farbrad, desto grösser ist der darstellbare Farbraum des Projektors. Das Farbrad dreht sich um sich selber, sodass nacheinander die Farben auf dem Farbrad entstehen. Das Bild entsteht schlussendlich durch das sogenannte DMD («Digital Mirror Device»). Das DMD ist ein Chip mit bis zu 2.2 Millionen kleinen Spiegeln darauf, wobei jeder Spiegel für einen einzelnen Bildpunkt auf dem finalen Bild steht. Wenn nun ein spezifischer Pixel die Farbe des Farbrades annehmen soll, wird der entsprechende Spiegel so ausgerichtet, dass der Lichtstrahl (ganz oder teilweise) auf die Linse gelenkt wird. Die Spiegel des DMD-Chips können bis zu 5000 Mal pro Sekunde geschaltet werden. Dies ist so schnell, dass das menschliche Auge die eigentlich aufeinanderfolgenden Farben wieder zu einem kompletten Bild zusammensetzt. [15]

DLP-Projektoren haben den Vorteil, dass sie einen sehr hohen Kontrast bieten, da sie schwarze Pixel gar nicht erst ausleuchten. Die extrem schnellen Schaltzeiten des DMD-Chips verhindern ein Nachziehen des Bildes. Nachteilig wirkt sich das konstruktionsbedingte Nacheinanderdarstellen der Farben aus. Bei schnellen Szenen kann sich dies im sogenannten «Regenbogeneffekt» äussern. Analog zum LCD-Projektor muss die Lampe nach etwa 4'000 Stunden ersetzt werden.

## LED

LED-Projektoren unterscheiden sich im Vergleich zu DLP- und LCD-Projektoren durch die Art der Lichtquelle. Anstatt einer Lampe kommen sogenannte LED's («Light Emitting Diode») zum Einsatz. Diese erzeugen die Farben Rot, Grün und Blau. Das Bild wird dann meistens mittels DLP-Chip erzeugt, teilweise kommen auch LCD's zum Einsatz. [16]

Je nach verwendeter Technologie kommen dieselben Vor- und Nachteile wie bei den oben genannten DLP- respektive LCD-Projektoren zum Tragen. Die LED's haben gegenüber Lampen den Vorteil, dass sie viel energieeffizienter sind und somit die erzeugte Abwärme sowie auch der Energiebedarf sinkt. Ausserdem weisen sie eine massiv höhere Lebensdauer auf (20'000 Stunden). Allerdings sind LED's bei weitem nicht so hell wie konventionelle Lampen. Ausserdem sind LED-Projektoren im Vergleich zu Lampenprojektoren eher teuer, das Preis-/Leistungsverhältnis also schlechter.

## Laser

Laser-Projektoren sind die modernsten Projektoren. Es gibt hier verschiedene Techniken wie beispielsweise Laser-Phosphor-Projektoren [17] oder RGB-Laser-Projektoren [18]. Bei den Laser-Phosphor-Projektoren besteht die Lichtquelle aus einem Laser-Array mit bläulichem Licht, welches mittels DLP-Chips oder LCD-Panels in ein Bild umgewandelt wird. RGB-Laser-Projektoren sind dem Hochleistungssegment und damit professionellen Anwendungen zuzuordnen. Diese Laser projizieren ebenfalls mittels LCD- oder DLP-Elementen.

Laserprojektoren erreichen sehr hohe Lichtstärken und eine lange Lebensdauer (bis zu 20'000 Stunden). Außerdem haben sie eine hohe Energieeffizienz und somit eine vergleichsweise geringe Wärmeentwicklung. Nachteile entstehen aus der Verwendung von DLP-Chips respektive LCD-Panels (bei den entsprechenden Projektoren diskutiert). Zudem sind sie im Vergleich zu LED-, LCD- und DLP-Projektoren teurer und weisen somit ein schlechtes Preis-/Leistungsverhältnis auf.

### 2.4.2 Weiterführende technische Aspekte

Neben den Vor- und Nachteilen verschiedener Anzeigeverfahren sind auch weiterführende technische Aspekte von Projektoren von Interesse.

#### Lichtstrom

Der Lichtstrom gibt die von einer Lichtquelle abgegebene Lichtmenge an. [19] Sie wird in Lumen  $lm$  angegeben. Bei jedem Projektor wird der Lichtstrom als Mass für die Helligkeit des Beamers angegeben. Höhere Werte resultieren in einem helleren Bild.

#### Beleuchtungsstärke

Die Beleuchtungsstärke setzt den Lichtstrom in ein Verhältnis zur Fläche. [19] Je weiter eine Lichtquelle von einem anzustrahlenden Objekt weg ist respektive je grösser die Fläche, die ausgeleuchtet werden soll, desto dunkler wird sie. Das Mass der Beleuchtungsstärke ist bei Projektoren aussagekräftiger als das Mass des Lichtstromes, da es die auszuleuchtende Fläche berücksichtigt. Die Einheit der Beleuchtungsstärke ist Lux  $lx$ . Je höher dieser Wert, desto heller die Fläche  $f$ . Die Beleuchtungsstärke wird mittels folgender Formel ermittelt:

$$lx = \frac{lm}{f(m^2)}$$

Für die Berechnungen der Beleuchtungsstärke in dieser Arbeit werden folgende Werte als Vergleich herangezogen: [20]

Umgebung	Lux
heller Sonnentag	100'000
Beleuchtung Fernsehstudio	1'000
Bürobeleuchtung	500
Strassenbeleuchtung	10
Kerze (ca. 1 Meter entfernt)	1
Mondlicht	0.25

Gemäss der Norm DIN 19045 sollte die von der Projektionseinheit erzeugte Beleuchtungsstärke  $plx$  mindestens 5 Mal so stark sein wie Beleuchtungsstärke des Umgebungslichts  $ulx$ . [21] Die benötigte Beleuchtungsstärke  $plx$  errechnet sich also wie folgt:

$$plx = ulx * 5$$

### **Projektionsverhältnis**

Die mechanischen Begebenheiten sowie die Optik eines Projektors ergeben ein bestimmtes Projektionsverhältnis. Das Projektionsverhältnis setzt die Projektionsdistanz in Relation zur Bildbreite. [22] Das Projektionsverhältnis  $pv$  errechnet sich bei gegebener Bildbreite  $b$  und Projektionsdistanz  $d$  wie folgt:

$$pv = \frac{b(cm)}{d(cm)}$$

### **Lens-Shift**

Unter Lens-Shift wird das optische Verschieben der Projektion verstanden. Dadurch kann die Projektion verschoben werden, ohne die Position des Projektoren selbst verändern zu müssen. [23] Diese Funktion macht eine aufwändige Optik im Projektor nötig. Der Vorteil gegenüber der digitalen Verschiebung, welche das Bild einfach in einem Teilbereich der Projektionsfläche darstellt, liegt darin, dass das verschobene Bild keine Verluste der Bildqualität (reduzierte Auflösung, Treppenartefakte, Kontrastverluste durch Pixel-Interpolation) aufweist. Diese Funktion ist aufgrund der speziellen Anforderungen an die Optik meist hochpreisigen, professionellen Projektoren vorbehalten.

### 3 Vorgehen / Methoden

Diese Arbeit unterscheidet sich in mehrfacher Hinsicht von anderen Software-Projekten. Die meisten Softwareprojekte leben vom Input des Kunden. Nicht nur zu Beginn eines Projektes, sondern auch während dessen Entwicklungslebenszyklus kommen normalerweise immer neue Anforderungen hinzu. In vorliegendem Fall verläuft das Projekt viel linearer. Zu Beginn wurden Ziele (siehe Kapitel 1.3) gesetzt sowie die Anforderungen definiert. Abgesehen von logistischen und administrativen Herausforderungen ändert sich an den Zielen nichts mehr.

Dieses Kapitel zeigt auf, wie wir an die Lösung der Aufgabenstellung herangegangen sind und wie diese Lösung umgesetzt wurde. Als integriertes Gesamtsystem kommen beim Orbitarium verschiedene Hardware- und Softwarekomponenten zum Einsatz. Zunächst analysieren wir die bestehende Lösung. Aus dieser Analyse folgt dann eine grobe Evaluation der Tools, welche die in der Analyse identifizierten Probleme lösen. Im Anschluss wird eine Vision der Lösung aus der Analyse der bestehenden Lösung sowie der Evaluation Tools vorgestellt. In einem nächsten Schritt wird dann die benötigte Hardware evaluiert. Nach der Beschreibung der Hardware folgt eine detaillierte Vorstellung der Software, welche die verschiedenen Herausforderungen löst und zu guter Letzt die eigens entwickelte Anwendung zur Steuerung des Gesamtsystems.

#### 3.1 Analyse und Evaluation

Im Gegensatz zu einem Projekt, in welchem eine neue Software geschrieben werden muss, verhält sich die Situation hier so, dass bereits eine funktionierende Lösung besteht. Da die Lösung jedoch bereits seit 10 Jahren in Betrieb ist, weist sie Mängel auf und muss daher überholt werden. Zunächst wird die funktionierende Lösung analysiert, um daraus die zu verwendenden Tools zu synthetisieren. Hierfür wurde das Orbitarium mit der bisherigen Lösung vor Ort begutachtet.

##### 3.1.1 Die bestehende Lösung

Bisher hing der Globus im Orbitarium des *Swiss Science Center*, dem Technorama. [1] In Kombination mit einer Video-Wand wurden auf dem Orbitarium Filme und Simulationen abgespielt. Der Raum wurde zu diesem Zweck stark abgedunkelt. Die Video-Wand ist komplett vom Orbitarium abgekoppelt.

##### Darstellung von Inhalten auf dem Orbitarium

Die Inhalte werden im Voraus aufbereitet. Zum Präsentationszeitpunkt werden dann die aufbereiteten Darstellungen mittels 2 Projektoren projiziert. Beide Projektoren sind an einem PC angeschlossen, welcher das vorab gerenderte Bild anzeigt. Dieses Bild besitzt eine Auflösung von 3840 x 1080 Pixel, was zwei nebeneinander gelegten Full-HD-Bildern entspricht. Die aufbereitete Azimutalprojektion (siehe Abschnitt 2.3.3) wurde vorgängig halbiert und nebeneinander wiederum zu einem Bild mit genannter Auflösung zusammengesetzt. Schlussendlich wurde das Bild über beide Bildschirme (in diesem Fall Beamer) angezeigt. Beide Beamer projizieren so jeweils eine Hälfte der Erdkugel. Mit zwei angeschlossenen Projektoren ist die Kapazität für Anzeigegeräte an der Grafikkarte jedoch ausgeschöpft. Um diesen PC nun trotzdem richtig konfigurieren zu können ist ein zweiter PC installiert, von welchem aus man mittels TeamViewer[24] auf den ersten PC zugreift und so die Konfigurationen vornehmen kann.

## Aufhängung

Die ganze Konstruktion (zu sehen in Abbildung 3 auf Seite 11) ist mittels Stahlträgern an der Decke montiert. Oberhalb des Projektorgehäuses ist eine in etwa flächengleiche Metallplatte montiert. Aus dieser kommen 4 Gewindeschrauben. Diese Schrauben werden in die Halterung nach oben hineingeschoben. Die Schrauben werden von Unterlagsscheiben und Muttern in Position gehalten. Der Sachverhalt ist in der Abbildung 7 dargestellt. Ebenfalls gut erkennbar ist der Befestigungspunkt des Projektorgehäuses an der Metallplatte in der Mitte (direkt unter dem weißen Kästchen). Die tragenden vier Metallstreben sind direkt an der Decke montiert. Die 4 Gewinde sind quadratisch angeordnet, haben einen Durchmesser von 11 Millimeter und eine Länge von etwa 20 Zentimeter. Siehe hierzu die Konstruktionszeichnung in Abbildung 8.



Abbildung 7: Aufhängungspunkt der Metallplatte an der Deckenhalterung. In der Mitte die Verbindung zum Projektorgehäuse.

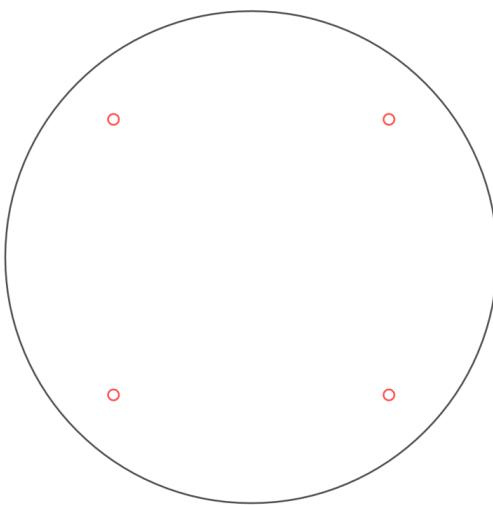


Abbildung 8: Schematische Darstellung der Aufsicht der Montageplatte. Rot die Montagegewinde.

Die Höhe des Raumes, in dem das Orbitarium im Technorama aufgebaut wurde, beträgt zwischen 5 und 6 Meter. Dies erlaubt einen geräumigen Abstand zur Decke für die Elektronik und trotzdem ca. 0.5 Meter Abstand zum Boden, gemessen vom tiefsten Punkt des Globus.

## Die Projektionsgeometrie

Die Projektionsmethode der Projektoren auf die Projektionskugel unterscheidet sich wesentlich von den bereits genannten Beispielen (beschrieben den Kapiteln 1.2.1 sowie 1.2.2 ab der Seite 8). Wie die Abbildung 3 (Seite 11) vermuten lässt, projizieren die Projektoren ihr Bild nicht direkt auf die Projektionskugel. Diese Projektionsgeometrie wird hier erläutert.

Unter dem Terminus Projektionsgeometrie subsummieren wir den Weg des Lichtes von der Linse der Projektoren bis hin zur Projektionskugel. In Abbildung 3 (Seite 11) ist das Orbitarium in der Realität dargestellt. Abbildung 9 hingegen zeigt das Orbitarium schematisch im Querschnitt. Zuoberst finden wir das «Projektorgehäuse» **1** mit der Montageplatte (hier nicht dargestellt). Im Projektorgehäuse befindet sich der «Projektorkäfig» **2** mit den Projektoren darin. Der «Projektorkäfig» hält die Projektoren an ihrem Platz und hat Markierungen, damit die Projektoren einfach eingesetzt werden können. Darunter befindet sich die Plexiglasröhre **3** gefolgt von der eigentlichen Projektionskugel **4**. Innerhalb des Globus befindet sich ein halbkugelförmiger Spiegel («Kugelspiegel») **5**. Der Pfeil **6** zeigt den Lichtfluss an.

Der Lichtfluss verläuft wie folgt: Zunächst wird das Licht vom Projektor auf einen  $5.5\text{cm} * 7\text{cm}$  grossen Spiegel («Projektorspiegel», siehe Abb. 9, blaue Fläche) projiziert. Dieser Spiegel ist um  $45^\circ$  ins Innere des Projektorkäfigs **2** abgedreht. Somit wird eine Reflexion der Projektion zur Seite erreicht. In der Mitte des Projektorkäfigs direkt über der Öffnung der Plexiglasröhre **3** befindet sich ein zweiter Spiegel («Kombinationsspiegel», siehe Abb. 9, orangene Fläche). Die Form dieses Spiegels erinnert an ein, auf dem Kopf stehendes Dreieck. Danach verlässt der Lichtstrom das Projektorgehäuse mit dem Projektorkäfig und wird durch die Plexiglasröhre hinunter auf den Kugelspiegel **5** und von dort aus schlussendlich auf die Innenwand der Projektionskugel **4** geleitet. Der Lichtfluss des hier diskutierten Projektors («Projektor 1») ist in Abbildung 9 rot dargestellt. Auf der gegenüberliegenden Seite findet sich die exakt selbe Konstruktion für den «Projektor 2» (der zugehörige Projektorspiegel ist weggelassen). Der Lichtfluss des Projektors 2 ist in Abbildung 9 grün dargestellt. Die beiden Beamer projizieren somit jeweils die Hälfte der Erdkugel (der Höhe nach halbiert). Der Lichtfluss vom Kugelspiegel zur Projektionskugel ist der Übersichtlichkeitshalber nicht dargestellt.

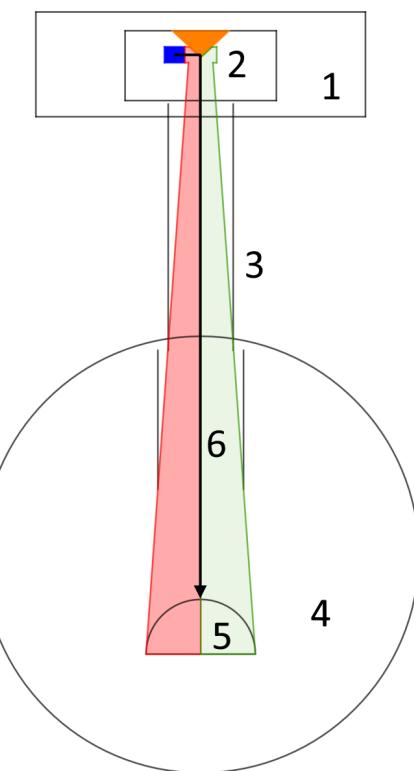


Abbildung 9: Querschnitt (mittig, massstabsgetreu) des Orbitariums mit Darstellung der Projektionsgeometrie.

Abbildung 10 zeigt den Projektionsspiegel von oben betrachtet sowie die Projektionsflächen der Projektoren. Die Projektionsfläche des Projektoren 1 ist rot dargestellt, die Projektionsfläche des Projektoren 2 ist grün eingefärbt. Ebenfalls ist ersichtlich, dass der Projektionsspiegel aufgrund seiner runden Beschaffenheit nicht das gesamte projizierte Bild auffangen kann. Der Verlust  $v$  an Bildpunkten beträgt bei jedem gegebenen Radius  $r$  in Prozent:

$$v = (-1) * \frac{r^2 * \pi}{(2 * r)^2} - 1 = 0.214601 \approx 21.56\%$$

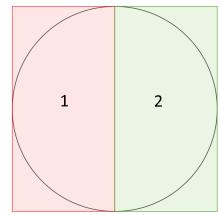


Abbildung 10: Aufsicht auf den Projektionsspiegel

### 3.1.2 Problemstellung und Risiken

Anhand obiger Entdeckungen werden die grössten Risiken abgeschätzt. diesen gehören unter anderem organisatorische, hardware-technische und administrative Risiken. Eine Auflistung der prägnantesten Risiken findet sich weiter unten. Die Abschätzung der Risiken bezüglich der Eintrittswahrscheinlichkeit und des Schadensausmasses ist in Abbildung 11 dargestellt. Hierbei sind die schwerwiegendsten Risiken oben rechts angesiedelt, wohingegen sich die weniger kritischen Risiken unten links befinden.

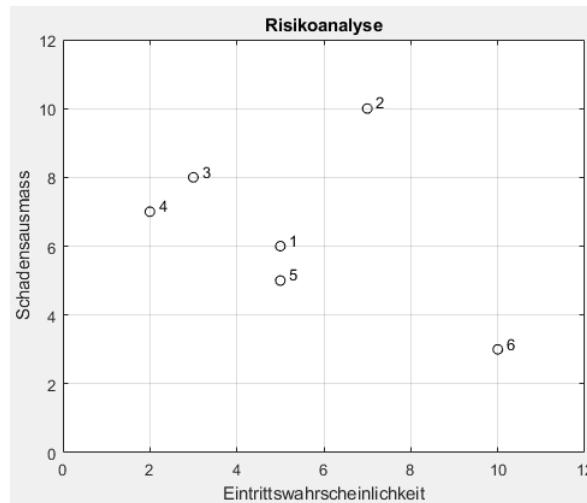


Abbildung 11: Risikoanalyse der Arbeit. Auf der X-Achse ist die Eintrittswahrscheinlichkeit von 0 - 10 abgebildet und auf der Y-Achse das Schadensausmass, ebenfalls von 0 - 10.

- Transport des Globus nicht möglich:** Der Transport an die ZHAW wäre für die Arbeit an dem Globus selber weniger ein Problem als schlussendlich für das Resultat der Arbeit.
- Keine Möglichkeit aktiv und praktisch zu testen:** Es besteht aus irgendwelchen Gründen keine Möglichkeit, die Software praktisch zu testen.
- Bestehende Hardware ungenügend:** Die Projektoren, der PC oder die Aufhängung im Technorama ist fehlerhaft oder defekt. Es müsste neben der Software noch die richtige Hardware gesucht und beschafft werden.

4. **Beschaffungskosten von Projektoren, PC und Aufhängung zu hoch:** Sollte die Hardware des Technoramas nicht im Preis inbegriffen sein oder müsste sie komplett neu ersetzt werden, könnte das Budget überstrapaziert werden.
5. **Aufhängung an der ZHAW nicht möglich:** Dies könnte aufgrund der Dimensionen der freien Räume passieren, oder dass keine passende Aufhängung gefunden werden kann.
6. **Fehlendes Know-How bezüglich verwendeter Technologien:** Dieses Risiko tritt mit hoher Wahrscheinlichkeit ein, kann jedoch durch Recherchetätigkeiten gemindert werden. Ein gewisses Restrisiko bleibt in Form eines überdimensionierten Scopes.

### 3.1.3 Wahl der Methode

Mit den errungenen Erkenntnissen aus der Analyse der Ist-Situation und der Risikoabschätzung gilt es zu entscheiden, inwiefern auf externes Know-How zurückgegriffen werden kann und welche Teile selbst entwickelt werden müssen.

#### Zusammenarbeit mit NOAA

Eine Zusammenarbeit mit NOAA würde eine Vielzahl von Vorteilen beinhalten. Wie in Kapitel 1.2.2 beschrieben, wird mit dem *Science on a Sphere*-System das *SOS Softwarepaket* mit einer Vielzahl an Steuerungs- und Darstellungssoftware mitgeliefert. Ein weiterer Pluspunkt wäre der grosse Fundus an bereits existierenden Animationen.

Wir hatten die Idee, in Zusammenarbeit mit der NOAA das *SOS Softwarepaket* zu erweitern, sodass nebst einer Bestrahlung von aussen auch die Bestrahlung von innen ermöglicht würde. Dies hätte zur Folge, das einiges an Entwicklung der Darstellungssoftware wegfielen und mehr Ressourcen in die Entwicklung von Animationen investiert werden könnten. Für die NOAA wäre diese Zusammenarbeit insofern interessant gewesen, als dass sie die Erzeugnisse dieser Arbeit in ihren Katalog integrieren hätte können.

Leider jedoch kam diese Kooperation nicht innert nützlicher Frist zustande. Die Kommunikation zwischen der in Amerika ansässigen Kontaktpersonen der NOAA war durch die Zweitverschiebung mühsam und beanspruchte viel Zeit. Ausserdem erhielten aufgrund des behördlichen Charakters der NOAA weder eine definitive Zu- oder Absage noch die Erlaubnis, auf ihre bestehende Software zuzugreifen.

#### Erweiterung / Verbesserung der Lösung des Technoramas

Ein weiterer Ansatz besteht darin, die Softwarekomponenten der Lösung des Technoramas weiterzuverwenden und nur die nötigen Veränderungen zur Steigerungen der Leistung vorzunehmen. Hierbei ist das Problem, dass die ursprüngliche Software zu kompakt und zu sehr auf den Endbenutzer zugeschnitten ist. Es stehen keine Schnittstellen zur Verfügung. Es ist zwar möglich, eigene Bilder umrechnen zu lassen und so einen anderen Output zu erhalten. Das Problem liegt darin, dass diese Umrechnung nicht in Echtzeit geschieht, womit ein zentrales Ziel der Arbeit verfehlt würde. Ausserdem ist die bestehende Lösung bezüglich der Ausgabe des Outputs auf mehrere Bildschirme sehr unflexibel und nicht sehr solide.

#### Zusammenarbeit mit Google

Eine Zusammenarbeit mit Google ist eine weitere Möglichkeit. Die Anwendung *Google Earth* scheint in eine ähnliche Richtung wie unsere Arbeit zu gehen. *Google Earth* projiziert ebenfalls eine Darstellung der Erde auf eine (simulierte) Kugeloberfläche. Des weiteren umfasst *Google Earth* auch

einen grossen Animationenkiosk und stellt überdies eine Schnittstelle zur Verfügung, um neue Animationen hinzuzufügen.

Im Gespräch mit Google stellte sich heraus, dass ebenfalls Interesse ihrerseits besteht, eine Zusammenarbeit mit uns einzugehen. Inmitten der Verhandlungen brach jedoch die COVID-19 Pandemie aus und unterband die potenzielle Zusammenarbeit abrupt.

## Eigene Lösung

In Ermangelung von Alternativen wurde schlussendlich entschieden, eine eigene Lösung zu entwickeln. Der Nachteil darin besteht in der beschränkten Zeit und dem Aufsetzen einer Umgebung beziehungsweise dem Entwickeln von Software, welche sich mehr auf die Steuerungsfunktionalität des Orbitariums konzentriert und weniger auf die Entwicklung eigener Darstellungen respektive Animationen. Genauere Erläuterungen zu dieser Steuerungssoftware finden sich im Kapitel 3.4.6. Der Vorteil ist hingegen, dass keine externen Beschränkungen und Richtlinien eingehalten werden müssen. Ebenfalls ist es so möglich, nach der eigentlichen Idee zu arbeiten und keine Kompromisse mit externen Stakeholdern eingehen zu müssen.

### 3.1.4 Evaluation der Tools

Grundsätzlich werden zwei Tools benötigt. Zum Einen ein Tool, welches die Umrechnung der Eingabe in Echtzeit durchführt und die Ausgabe des Bildes übernimmt und zum Anderen brauchen wir eine Software, welche in Echtzeit und interaktiv den gewünschten Input an das erste Tool weiterreicht.

Hierbei sehen die Anforderungen so aus:

- Die Tools müssen möglichst performant laufen.
- Die Auswahl des Inputs muss interaktiv und dynamisch geschehen.
- Die Umrechnung der Darstellung muss in Echtzeit und unabhängig vom Input geschehen.
- Die Rechenarbeit soll wenn möglich auf der Grafikkarte geschehen.
- Die Tools sollten ein qualitativ annehmbares Resultat erzeugen bezüglich Auflösung und Bildrate.
- Die Tools sollten kostenlos sein oder einen niedrigen Kostenpunkt besitzen.
- Die Schnittstellen der Tools müssen kompatibel zueinander sein.

## Unity

Es kristallisierte sich sehr schnell heraus, dass sich für die Berechnung und Verarbeitung des Inputs Unity besonders gut eignet. Die Vorteile von Unity sind zahlreich und ausserdem wurde es bereits in einem besuchten Modul als dediziertes Tool verwendet. Mit Unity ist es möglich die Berechnungen in Echtzeit auf der GPU laufen zu lassen, was im Gegenzug mehr Rechenleistung für andere Prozesse des Computers frei lässt. Ebenfalls vielversprechend sind die eigens erstellbaren Shader von Unity, welche dazu dienen, das Bild beliebig zu transformieren. Mehr dazu im Kapitel 3.4.4. Zu guter Letzt ist nicht zu vernachlässigen, dass Unity für Studenten kostenlos verfügbar ist.

## ManyCam

Die Wahl des zweiten Tools gestaltet sich anspruchsvoller. Es stellt sich die Frage, wie der Input in Unity aussehen sollte. Recherchen zeigten, dass Unity, neben anderen sehr nützlichen Features, Unterstützung für Webcam-Feeds bietet. Dies bedeutet, dass die Darstellung eines Bildschirmes direkt und in Echtzeit in Unity eingespeist werden kann. Somit ergibt sich, dass eine Bildschirmaufnahmesoftware verwendet werden soll. Dies ermöglicht den gewünschten interaktiven und dynamischen Wechsel des Inputs.

Die Wahl fiel schlussendlich auf die Software *ManyCam*. [25] ManyCam unterstützt in der Vollversion als eines der wenigen Tools in diesem Bereich eine 4K-Auflösung. Ein weiterer Vorteil dieses Tools ist, dass der Input jeweils nicht auf dem Bildschirm angezeigt werden muss, um aufgenommen werden zu können. So ist es beispielsweise möglich, eine Webseite in minimiertem Zustand geöffnet zu haben und trotzdem nur den Inhalt der Webseite anstelle des Desktops via ManyCam aufzunehmen. In unserem Setup (siehe nächstes Kapitel) kommt dies sehr gelegen, da nun ein Bildschirm als Input-Quelle und als Konfigurationsschnittstelle dient.

## 3.2 Vision der Lösung

Ausgehend von der Aufgabenstellung, der Analyse der bisherigen Lösung sowie der Evaluation geeigneter Tools haben wir eine Vision unseres Systems entwickelt.

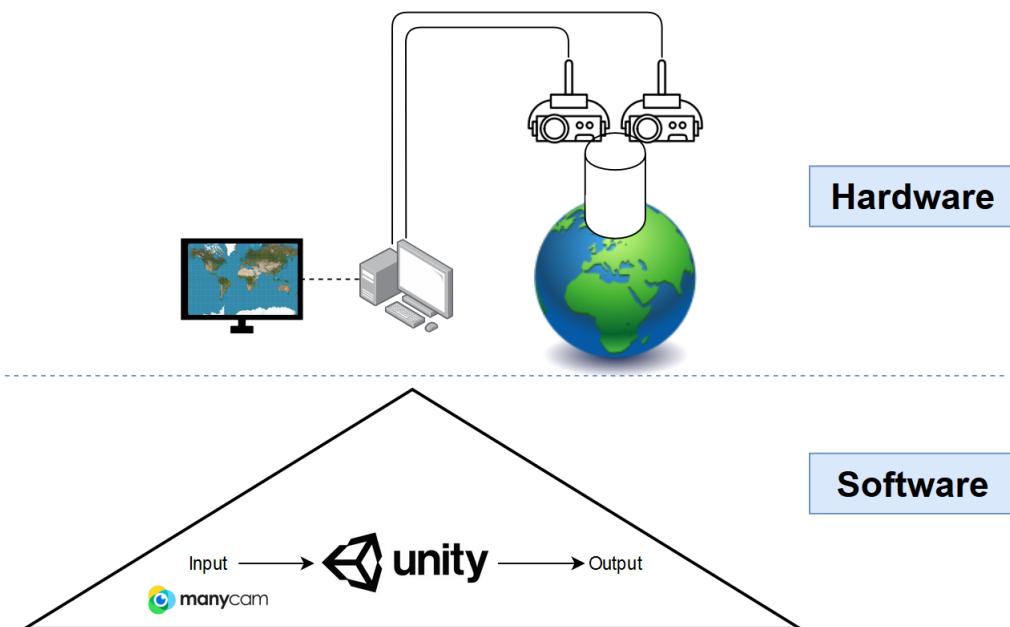


Abbildung 12: Die Vision des Gesamtsystems

Abbildung 12 zeigt unsere Vision des Gesamtsystems. Grundsätzlich soll mittels ManyCam ein geeigneter Input an Unity geliefert werden. Unity wiederum rechnet den Input um und sendet ihn über die Ausgänge an die beiden Projektoren. Die Umsetzung der Vision erfordert Hard- sowie Software. Diese beiden Teilprobleme gliedern sich in weitere Teilprobleme auf, die es zu lösen gilt. Im Kapitel 3.3 wird die Hardware abgehandelt, im Kapitel 3.4 wird einerseits die Software von Drittanbietern diskutiert, welche für den Betrieb des Orbitariums nötig ist und andererseits die selbst geschriebene Software erläutert.

### 3.3 Hardware

#### 3.3.1 Projektoren

Wie unter den Abbildungen 9 (Seite 20) sowie 12 gezeigt und im Kapitel 3.1.1 diskutiert, benötigt das Orbitarium 2 Projektoren. Im Orbitarium waren 2 Projektoren des Typs «Projectiondesign FL32» [26] verbaut. Mit jeweils rund 25'000 Stunden Betriebsdauer sind diese Geräte am Ende ihrer Lebensdauer angelangt. Zusätzlich ist der eine Projektor nicht funktionsfähig und reparaturbedürftig. Die Projektoren mussten demnach ersetzt werden. Im Folgenden werden die Charakteristika passender Projektoren errechnet.

#### Das Projektionsverhältnis

Das Projektionsverhältnis errechnet sich aus der Bildbreite und der Projektionsdistanz. Die Projektionsdistanz ist der Weg des Lichtes von der Linse des Projektors bis hin zum Kugelspiegel und errechnet sich wie folgt:

Teilstrecke	Distanz (in cm)
Projektorlinse $\Rightarrow$ Projektorspiegel	8
Projektorspiegel $\Rightarrow$ Kombinationsspiegel	16.6
Kombinationsspiegel $\Rightarrow$ Kugelspiegel	189.08 - 175.53
<b>Total</b>	200.13 - 213.68

Da die Projektionsdistanz aufgrund des halbförmigen Spiegels nicht immer gleich ist und wir die Distanz nicht immer genau abmessen konnten, rechnen wir mit einer Sicherheitsmarge. Wir gehen von einer Projektionsdistanz von *190 - 230cm* aus.

Die Breite der Bildfläche pro Projektor ergibt sich aus dem Durchmesser des Kugelspiegel (vgl. Abb. 10) und beträgt *36cm*. Daraus errechnet sich das Projektionsverhältnis *pv* (vgl. Kapitel 2.4):

$$pv = \frac{190 - 230}{36} = 5.05:1 - 6.11:1$$

#### Der Lichtstrom

Der Lichtstrom der Projektoren muss genug stark sein, um die Projektionskugel sauber auszuleuchten. Da bis dato noch unklar ist, wo das Orbitarium an der ZHAW zu stehen kommt, gehen wir davon aus, dass das Umgebungslicht eine Beleuchtungsstärke von *500 Lux*, also das Äquivalent einer Bürobeleuchtung beträgt.

Der Durchmesser *d* der Projektionskugel beträgt *150cm*, woraus die Oberfläche *o* der Kugel folgt:

$$o = 150^2 * \pi \approx 70686cm^2$$

Jeder Projektor bestrahlt nur die Hälfte der Kugel. Pro Projektor betragen die auszuleuchtenden Flächen  $f_1, f_2$  somit

$$f_1, f_2 = \frac{70685.834 \text{ cm}^2}{2} \approx 35342 \text{ cm}^2$$

*Anmerkung: Obschon nicht die ganze Projektionskugel bestrahlt wird (der Bereich unter dem Projektionsspiegel wird nicht angestrahlt und fällt somit als Projektionsfläche weg), nehmen wir die ganze Projektionskugel als Referenz. Es besteht somit auch hier eine Reserve.*

Aus der Formel für die Beleuchtungsstärke  $lx$  (siehe Kapitel 2.4) folgt der benötigte Lichtstrom  $lm$  bei gegebener Fläche  $f$  (in  $\text{m}^2$ ). Zu beachten ist außerdem, dass die Beleuchtungsstärke das 5-fache des Umgebungslichtes betragen muss.

$$lm = lx * f = (500 * 5) * 3.534 \approx 8835 lm$$

### Seitenverhältnis und Auflösung

Unter dem Seitenverhältnis eines Projektors versteht man das Verhältnis von Bildbreite zu Bildhöhe. Es wird mit  $n : m$  angegeben, wobei  $m$  für die Bildbreite und  $n$  für die Bildhöhe steht. Wie Abbildung 10 zeigt, muss im Orbitarium mit beiden Projektoren ein quadratisches Bild erzeugt werden. Also muss mindestens  $2n = m$  für die beiden Projektoren gelten. Allerdings müssen bei einem solchen Setting die Projektionen exakt nebeneinanderliegen, damit sich kein störender schwarzer Balken vertikal über die Projektionskugel ausbreitet. Deshalb kommen nur Projektoren infrage, für welche  $2n > m$  gilt. Unter der Berücksichtigung aktueller Projektoren kommen folgende Projektionsverhältnisse infrage:

- 16 : 9
- 16 : 10

Um eine möglichst scharfe Darstellung des Inhaltes auf der Projektionskugel zu gewährleisten, ist eine hohe Auflösung von Vorteil. Deshalb schlagen wir eine Auflösung von *3840 x 2160 Pixel* vor, was einem Seitenverhältnis von *16 : 9* entspricht. Somit bleiben  $2 * 2160 - 3840 = 480 \text{ Pixel}$  für eine Überlappung der beiden Projektionen übrig.

## Linsenposition, Lens-Shift, Projektordimensionen und Technologie

Ein Projektor muss die Linse an der exakt richtigen Position haben sowie in den Projektorkäfig passen. Abbildung 13 zeigt die schematische Darstellung des Projektorkäfigs. Ein Projektor darf maximal die folgenden Dimensionen aufweisen:  $54 \times 24 \times 32\text{cm}$  (Breite **1** x Höhe **3** x Tiefe **2**). Außerdem muss die Linse exakt auf den Projektorspiegel passen (siehe Abb. 13, blaue Fläche). Der Projektorspiegel ist  $13.5\text{cm}$  von der linken Beschränkung **5** und  $13\text{cm}$  vom Boden des Projektorkäfigs **4** entfernt. Um das projizierte Bild ohne das Verschieben des Projektors bewegen zu können, muss das Gerät Lens-Shift unterstützen (siehe Kapitel 2.4). Aufgrund dem hohen geforderten Lichtstrom schlagen wir ausserdem einen Laser-Projektor vor.

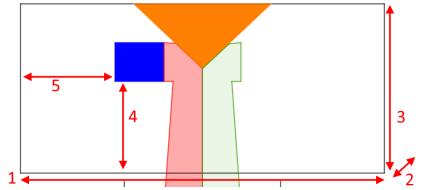


Abbildung 13: Ansicht des Projektorkäfigs

## Beschaffung und Gegenüberstellung

Um passende Projektoren beschaffen zu können, wurden diverse Firmen angefragt. Nach einiger Zeit machten wir mit der Firma Lang Baranday [27] Bekanntschaft. Sie offerierten uns die Möglichkeit, verschiedene Projektoren zur Evaluation zu mieten und auch zu kaufen. Unter anderen schlug die Firma das Gerät Barco FL40-4K [28] vor.

Leider kamen wir aufgrund des immer enger werdenden Zeitplans nicht mehr dazu, verschiedene Projektoren zu evaluieren. Stattdessen kommen eigentlich vom Technorama nur als kurzfristigen Ersatz für das Orbitarium angedachte Projektoren des Typs projectiondesign FL32 [26] zum Einsatz. Nachstehende Tabelle vergleicht die evaluierten Charakteristiken der Projektoren mit den effektiven:

Merkmal	evaluiert	projectiondesign FL32
Projektionsverhältnis	5.05:1 - 6.11:1	3.80:1 - 6.50:1
Lichtstrom (lm)	8835	600
Seitenverhältnis	16 : 9	16 : 9
Auflösung (Pixel)	3840 x 2160	1920 x 1080
Lens-Shift	Ja	Ja
maximale Abmessungen (B x H x T, in cm)	54 x 24 x 32	51 x 22 x 32
Linsenposition (von links, in cm)	13.5	keine Angabe, passend
Linsenposition (von unten, in cm)	13	keine Angabe, passend
Technologie	Laser	LED

### 3.3.2 PC

Die Entscheidung, auf hohe Auflösungen zu setzen, erfordert leistungsstarke Hardware. Insgesamt müssen  $3'840 * 2'160 * 2 \approx 16.6 \text{ Mio}$  Bildpunkte pro Frame berechnet werden. Deshalb setzen wir auf aktuelle Hardware. Da die Berechnungen von Unity hauptsächlich auf der Grafikkarte laufen, ist die Prozessorleistung sekundär. Wir haben uns deshalb entschieden auf die leistungsstarken nVidia-Karte zu setzen, die RTX 2080 TI [29]. Als Workstation kommt ein Dell Precision T5810 [30] zum Einsatz.

## 3.4 Software

Das Orbitarium benötigt nebst der unter Kapitel 3.3 beschriebenen Hardware auch ein geeignetes Softwarepaket, um die Projektoren mit geeignetem Bildmaterial zu beliefern. Im Kapitel 3.1.4 wurden bereits die Tools Unity und ManyCam evaluiert. In diesem Kapitel wird das Zusammenspiel dieser beiden Tools sowie weiterer Anwendungen, welche spezifische Aufgaben erfüllen, diskutiert. Das Kapitel 3.4.6 erklärt schlussendlich die selbst geschriebene Unity-Applikation, welche alle Komponenten integriert und die Animationen startet.

### 3.4.1 Die Darstellung der Animationen

Die erste Herausforderung stellt sich in der Darstellung der Erde. Nachdem die Kooperation mit Anbietern von Karten- und Animationsmaterial (siehe 3.1.3) gescheitert war, wurden andere Wege gesucht. Schlussendlich haben wir uns entschieden, einen Browser für die Darstellung der Erde zu verwenden. Dies erlaubt es, praktisch beliebigen Inhalt darzustellen. Im Internet gibt es bereits viele Webseiten mit interessanten Animationen. Beispiele dafür sind:

- Flightradar24 [31], eine Live-Darstellung aller kommerziellen Flüge weltweit
- Windy [32], eine Live-Darstellung der aktuellen Wind- und Wetterverhältnisse
- Google Maps [33], eine Darstellung der Welt mit diversen Interaktionsmöglichkeiten

### Google Maps zur Entwicklung eigener Animationen

Google Maps ist eine Internet-Plattform, welche die Erde unter Verwendung der Merkator-Projektion (siehe Kapitel 2.3.2) darstellt. Nebst diversen Möglichkeiten auf der Plattform selbst bietet Google mittels der «Keyhole Markup Language (KML)» [34] die Möglichkeit, die Karte zu bearbeiten. Hierfür wird ein Account benötigt, um auf die Google Maps API zugreifen zu können, welche die Bearbeitung der Karte schlussendlich erlaubt. Mittels einem API-Key wird die Map initialisiert und ein Java-Skript von den Google-Servern geladen, mit welchem die Veränderungen an der Karte schlussendlich vorgenommen werden können. Abbildung 14 zeigt eine einfache, mittels KML generierte Ansicht von Google Maps. Sie zeigt alle Flüge auf dem Gebiet, welches vom Open Sky Network [35] derzeit getrackt wird. Der Code für dieses Beispiel kann im Anhang unter A.2.7 sowie A.2.8 eingesehen werden. Ebenfalls ist die verwendete Merkatorprojektion gut zu sehen, da Grönland in etwa gleich gross wie Afrika erscheint, obwohl Grönland etwa 14 Mal kleiner als Afrika ist. Nebst einfachen Punkten, in der KML-Terminologie «Marker» genannt, können auch ganze Flächen als «Overlays» gezeichnet und viele weitere Anpassungen vorgenommen werden. Siehe hierzu die KML-Dokumentation. [36]

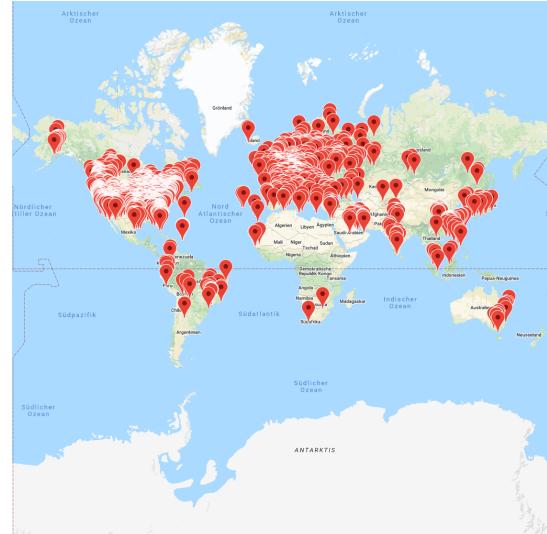


Abbildung 14: Google Maps mit eigenem KML-Overlay

### 3.4.2 Die Quelle des Inhalts für ManyCam

Nachdem die Frage geklärt ist, wie die Erde respektive die Animationen dargestellt werden sollen, muss geklärt werden, wie dieser Input Unity zugeführt werden soll (siehe auch Abbildung 12 für die Vision des Systems). Als Tool für diese Aufgabe wurde bereits ManyCam evaluiert (siehe 3.4.3). Es wurden verschiedene Varianten geprüft, um den Browser darzustellen und mit ManyCam aufzunehmen.

#### Separes Fenster

Die einfachste Variante zeigt den Browser in einem separaten Fenster und ManyCam nimmt den Browser auf. Die Vorteile dieser Lösung sind die folgenden:

- + Einfaches Setup
- + Erfordert keine weiteren Anpassungen

Dem gegenüber stehen die folgenden Nachteile:

- Das Browserfenster kann nicht im Vollbildmodus laufen, da ansonsten die Steuerkonsole verdeckt wird.
- ManyCam wird nicht nur die Animation aufnehmen, sondern auch noch das Browser-Fenster (Adressleiste, Steuerknöpfe sowie die Menü-Leiste).
- Das Fenster kann nicht grösser werden als die Auflösung des verwendeten Monitors, was bei grossen benötigten Auflösungen für die Projektoren zum Problem werden kann.

#### Separater Monitor

Die nächste Variante umfasst den Anschluss eines zusätzlichen, externen Monitors. Der Browser mit der Animation läuft auf diesem externen Monitor und wird von dort aus von ManyCam aufgenommen.

- + Das Browserfenster läuft im Vollbildmodus und kann ohne störende Fenster-Elemente aufgenommen werden.
- + Die Auflösung des Monitors kann so gewählt werden, dass sie grösser ist als die der Projektoren.

Dem gegenüber stehen die folgenden Nachteile:

- Zusätzliche Hardwarekosten durch den separaten Monitor
- Bedarf an einem zusätzlichen Monitoranschluss an der Grafikkarte
- Bei abweichenden Auflösungen zwischen Projektoren und separaten Monitor können durch die Pixel-Interpolation störende Artefakte im Ausgabebild erscheinen.

#### Virtueller Monitor

Bei der Darstellung des Browserfensters auf einem virtuellen Monitor hat folgende Vorteile:

- + Die Grösse des virtuellen Monitors kann flexibel den Begebenheiten (Auflösung der Projektoren, Anpassungen am an Unity zu liefernden Bildmaterial) angepasst werden.
- + Das Browserfenster kann im Vollbildmodus laufen.

Dem gegenüber stehen die folgenden Nachteile:

- Die Animation wird nirgendwo dargestellt, was die Fehlerbehebung schwierig macht
- Eine weitere Applikation wird benötigt

Nach eingehender Prüfung aller Varianten haben wir uns entschieden, die Variante mit dem virtuellen Monitor zu realisieren.

## Spacedesk Virtual Display Driver

Der Spacedesk Virtual Display Driver ist Teil der «spacedesk Multi Monitor App». [37] Sie kann dazu verwendet werden, um einen zusätzlichen Monitor zu emulieren und über eine spezielle App oder auch den Browser auf diesen zusätzlichen Monitor zuzugreifen. Hierzu installiert man den «spacedesk DRIVER for PRIMARY Machine». Nach der Installation dieses Treibers kann man mit einem Browser auf die Seite <http://viewer.spacedesk.net/> navigieren.

Nach der Eingabe der eigenen IP-Adresse und der Bestätigung erscheint ein zusätzlicher Monitor in den Systemeinstellungen. Abbildung 15 zeigt die Windows-Anzeigeeinstellungen vor und nach der Aktivierung des virtuellen Monitors. Abbildung 16 zeigt die Einstellungsmöglichkeiten des virtuellen Monitors. Hier kann unter anderem die Auflösung eingestellt werden. In den Windows-Anzeigeeinstellungen wird wiederum festgelegt, ob der zusätzliche Monitor den Inhalt der bereits bestehenden Monitore erweitern oder spiegeln soll.



Abbildung 15: Windows-Anzeigeeinstellungen: Links vor, rechts nach der Aktivierung des virtuellen Monitors

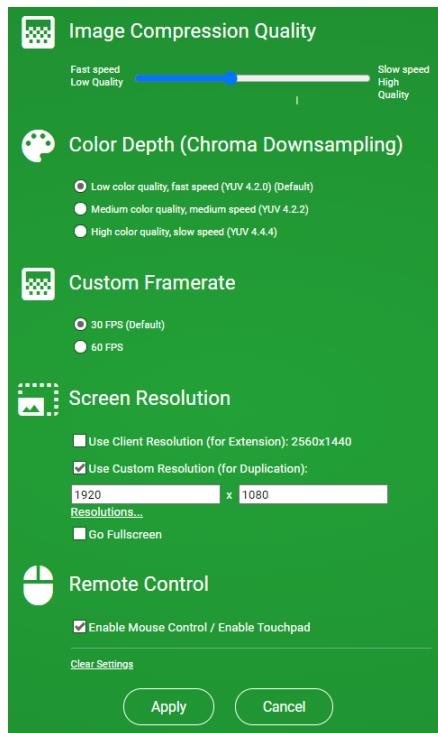


Abbildung 16: Einstellungen virtueller Monitor

### 3.4.3 ManyCam und Stream in Unity

ManyCam ist eine Software für die Aufnahme und das Streaming verschiedenster Quellen, so auch ein App-Fenster oder ein ganzer Bildschirm. [25]. Die Software wurde bereits evaluiert und unter 3.1.4 diskutiert. ManyCam installiert eine virtuelle Webcam auf dem System, auf welche der selektierte Input «gestreamt» wird. Dieser Stream wiederum kann von Unity mittels der Klasse «WebCamTexture» empfangen und beliebig weiterverwendet werden. [38] Somit ist die Animation nun in Unity als Input verfügbar.

### 3.4.4 Shaderlösung mit Unity

Um den Input richtig umrechnen zu können, wird von Unity ein Shader verwendet. Wie in Kapitel 2.2.1 erklärt beinhaltet Shader die mathematische Anweisung, wie die einzelnen Pixel dargestellt werden sollen.

Intuitiv könnte man denken, dass ein Shader das Input-Bild als Quelle nimmt und anhand der Umrechnungsanweisung jeden Pixel auswertet und an der richtigen Stelle im resultierenden Bild ausgibt. Unity nimmt jedoch den umgekehrten Weg. Unity geht jeden Pixel im (noch nicht dargestellten Resultat) durch, führt die Umrechnungen aus und evaluiert im Quell-Bild den resultierenden Pixel.

**Beispiel:** Anstatt im Quell-Bild den Pixel an den Koordinaten (1, 1) zu nehmen, diesen umzurechnen und schliesslich an den Koordinaten (15, 73) im resultierenden Bild auszugeben, nimmt Unity das noch nicht existierende, umgerechnete Bild und eruiert, was an der Stelle (1, 1) darzustellen ist. Anhand der Umrechnung erhält man dann ein Set von Koordinaten, z.B. (32, 10). An diesen Koordinaten wird das Quell-Bild evaluiert und so der darzustellende Pixel an der Stelle (1, 1) ermittelt.

Diese Erkenntnis ist für die Berechnung der Transformation grundlegend. Anstatt vorwärts muss rückwärts gerechnet werden. Für den Fall der Azimutalprojektion heisst das konkret, dass von der Azimutalprojektion über die Equirektangularprojektion zur Merkatorprojektion gerechnet werden muss, sofern der Input eine Merkatorprojektion ist. Hierbei nimmt man die Längen- und Breitenwinkel ( $\varphi, \theta$ ) als Zwischenschritt zwischen Azimutal- und Merkatorprojektion.

Die Formel, um von den Polarkoordinaten  $(R, \Theta)$  der Azimutalprojektion auf die Kugelkoordinaten  $(\varphi, \theta)$  zu schliessen ist hier abgebildet, wobei  $R$  den Radius und  $\Theta$  den Öffnungswinkel bezeichnet:

$$(\varphi, \theta) = \left( 2 \arccos \frac{R}{2}, \Theta \right)$$

Mit diesen Kugelkoordinaten und folgender Formeln kann wiederum rückwärts auf die planaren X/Y Koordinaten der Merkatorprojektion geschlossen werden:

$$x = R(\theta - \theta_0), \quad y = R \ln \left[ \tan \left( \frac{\pi}{4} + \frac{\varphi}{2} \right) \right]$$

Hierbei ist weiter zu beachten, dass in unserem Fall mit  $R = 1$  gerechnet wird und dass  $\theta_0 = 0$  ist. Somit bleibt in der obigen Formel für  $x$  lediglich  $\theta$ , was dem eigentlichen Winkel entspricht ( $\theta$  in Radianten).

Wie auffällt, behandelt man in diesen Umrechnungen stets die Azimutal- und die Merkatorprojektion, nicht aber die Equirektangularprojektion. Dies ist dadurch begründet, dass diese Formeln immer die Equirektangularprojektion als Ausgangslage nehmen, aus Gründen, die in Kapitel 2.3.1 behandelt werden. Man rechnet also implizit in die Equirektangularprojektion um, wenn man den Zwischenschritt über die Kugelkoordinaten nimmt. Alternativ könnte auch direkt von der Azimutalprojektion in die Merkatorprojektion gerechnet werden.

Der implementierte Shader und die umgesetzten Berechnungen sind im Anhang A.2.6 auf Seite 61 ersichtlich.

### 3.4.5 Ausgabe auf mehreren Bildschirmen

Jeder Projektor muss die Hälfte der Azimutalprojektion ausgeben (siehe Abb. 12). Dementsprechend muss diese Aufgabe mit Unity gelöst werden. In Abbildung 17 ist der Sachverhalt dargestellt. Das von Unity gerenderte und transformierte Bild ist grau eingefärbt. Die beiden Kameras, welche jeweils die Ausgabe für einen Projektoren übernehmen, sind rot und blau eingefärbt. Die beiden Kameras sind so ausgerichtet, dass die eine Kamera die obere Hälfte des Bildes aufzeichnet und die andere Kamera die untere.

In den Konfigurationseinstellungen der Kameras findet man ein Drop-Down Menü, genannt *Target Display*, welches eine Auswahl von nummerierten Displays zur Verfügung stellt (die Nummerierung reicht von 1 - 8). Diese Displaynummern stimmen exakt mit denjenigen überein, welche bei den Windows-Anzeigeeinstellungen zur Identifizierung der jeweiligen Displays hinterlegt sind. Dies führt dazu, dass Unity die richtigen Bildschirme beziehungsweise Projektoren ansteuert und für die Ausgabe auswählt.

Wenn man die bestehende Lösung (beschrieben in Abschnitt 3.1.1) als Referenz nimmt und mit der Art und Weise vergleicht, wie die Projektion eines Bildes auf zwei Projektoren erreicht wurde, sieht man was für einen enormen Vorteil Unity und die damit verbundene Flexibilität zur Ausgabe bietet.



Abbildung 17: Sehr rudimentäre Darstellung der Canvas und der beiden Kameras.

### 3.4.6 Das Orbitarium Control Panel

Aufgrund obiger Überlegungen und Tools haben wir den Prototypen einer Steuerkonsole (das «Orbitarium Control Panel») entworfen, welches die individuellen Komponenten anspricht und aktiviert, den Input aus ManyCam in Unity streamt, den Input umrechnet und schlussendlich an die beiden Projektoren weiterreicht. Die von uns entwickelte Software deckt die benötigte Funktionalität ab, ist aber erst im Stadium eines Prototypen. Welche Bestandteile funktionieren und welche nicht, ist im Kapitel 4 beschrieben. Dieses Kapitel soll einen Überblick über das User Interface des Orbitarium Control Panels geben, wohingegen das Kapitel 3.4.7 die Implementierung der Funktionalitäten behandelt. Abbildung 18 auf der nächsten Seite zeigt die Benutzeroberfläche des Orbitarium Control Panel. Es ist in 3 wesentliche Abschnitte gegliedert, welche farblich hervorgehoben sind und folgend weiter erläutert werden.

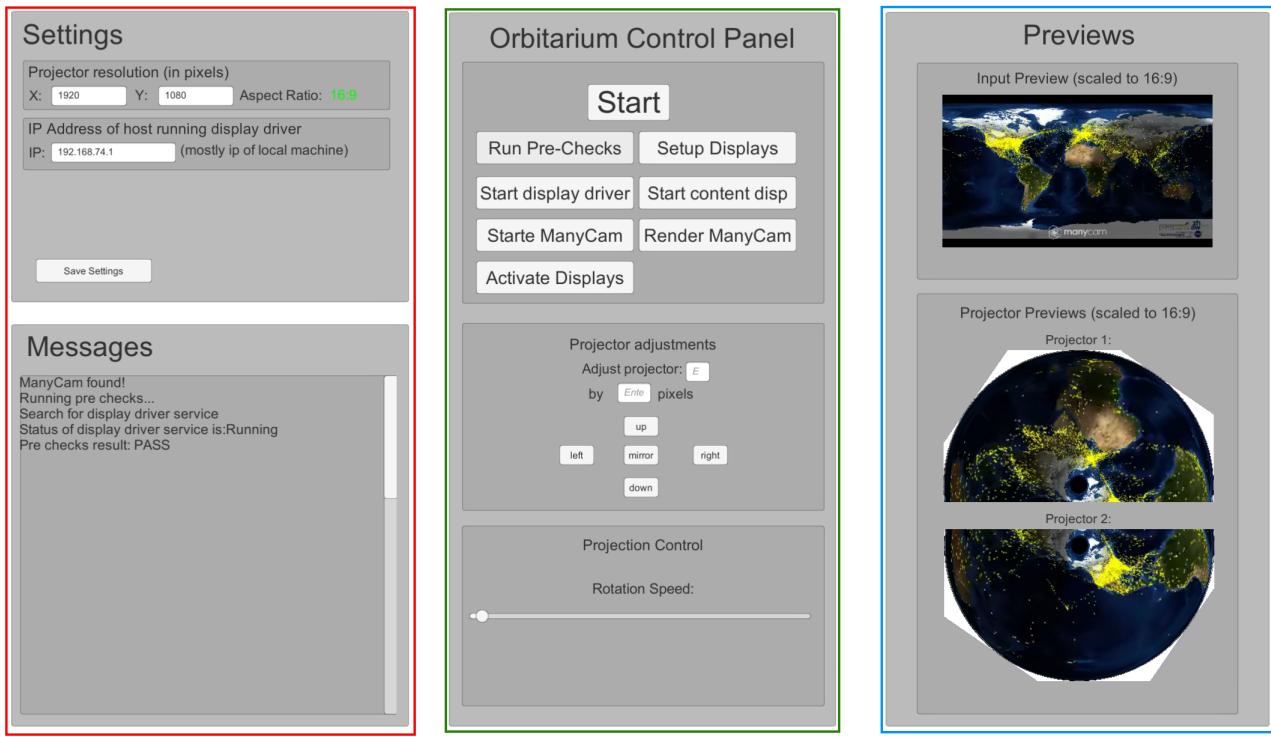


Abbildung 18: Orbitalium Control Panel. Die einzelnen Bestandteile sind farblich hervorgehoben.

## Settings

Im Settings-Panel (siehe Abb. 18, rot umrandeter Bereich) werden alle erforderlichen Einstellungen vorgenommen. Zum Einen wird hier die Auflösung für die Projektoren (und somit auch die Eingangs-Auflösung) festgelegt und zum Anderen wird die IP-Adresse des eigenen Rechners einge tragen. Damit kann der Treiber des virtuellen Displays (siehe Kapitel 3.4.2) über die Web-Oberfläche angesprochen werden. Im Panel «Messages» sind die Ausgaben der Anwendung zu finden. In der Abbildung sind die Ausgaben der «Pre checks» zu finden, welche verifizieren, ob alle benötigten Einstellungen vorgenommen wurden und die Applikation lauffähig ist.

## Orbitarium Control Panel

Das Orbitarium Control-Panel (siehe Abb. 18, grün umrandeter Bereich) ist das Herzstück der Steuerkonsole. Hier sind die verschiedenen Funktionalitäten zur direkten Steuerung der Anwendung untergebracht. Im oberen Teil sind alle Funktionen zum Starten der Anwendung untergebracht. Folgende Funktionalitäten wurden implementiert:

- Run Pre-Checks: Diese Funktion führt Überprüfungen durch, um sicherzustellen, dass die Anwendung lauffähig ist.
- Setup displays: Diese Funktionalität setzt mittels der in den Settings definierten Auflösungen die Unity-internen Verarbeitungskomponenten (Umrechnungsshader, Ausgabe-Kameras) auf.
- Start display driver: Hiermit wird der Treiber für das virtuelle Display gestartet und die Auflösung für selbiges gesetzt.
- Start content disp: Dieser Button aktiviert den Browser für die Darstellung der Inhalte und setzt ihn im Vollbild-Modus auf das virtuelle Display.

- Starte ManyCam: Diese Schaltfläche startet ManyCam.
- Render ManyCam: Hiermit wird der Webcam-Feed von ManyCam auf die benötigten Komponenten zur Umrechnung der Projektion abgebildet.
- Activate Displays: Diese Funktion aktiviert die externen Displays (in diesem Fall die Projektoren).

Im mittleren Panel «Projector adjustments» kann die Projektion von beiden Projektoren angepasst werden. Wie unter Abbildung 17 gezeigt und unter Kapitel 3.3.1, Abschnitt Seitenverhältnis und Auflösung erklärt, müssen sich die Projektionen überlappen. Um den genauen Überlappungsbereich zu finden, kann mittels der Buttons «up», «down», «left» und «right» die Projektion entsprechend verschoben werden. Dies ist unter Angabe der zu verschiebenden Pixel möglich, was eine präzise Anpassung an die Begebenheiten ermöglicht. Durch den Button «mirror» kann die Projektion ausserdem horizontal gespiegelt werden. Dies ist bei einem der beiden Projektoren nötig.

Zu guter Letzt findet sich im untersten Panel noch die «Projection control», welche es ermöglicht, die Rotationsgeschwindigkeit der Darstellung auf der Projektionskugel einzustellen.

## Previews

Das Previews-Panel (siehe Abb. 18, blau umrandeter Bereich) stellt dem Anwender eine Vorschau des Ergebnisses zur Verfügung. Im oberen Teil «Input Preview» findet sich eine Darstellung des Inputs von ManyCam, im unteren Teil «Projector Previews» wird die Vorschau der Ausgabe auf die beiden Projektoren dargestellt.

### 3.4.7 Implementierung des Orbitarium Control Panel

Dieses Kapitel widmet sich der konkreten Implementierungen der wichtigen Aspekte der unter Kapitel 3.4.6 andiskutierten Funktionalitäten. Unity verwendet die Programmiersprache C#, um das Verhalten von Objekten zu steuern, weswegen die Funktionalitäten in C# implementiert sind. Es werden nur die wichtigsten Aspekte der Implementierung behandelt. Nebenfunktionalitäten sind in den Erklärungen ausgelassen, aber dennoch im Anhang unter A.2 gelistet.

#### Klasse ApplicationManager

Die Klasse ApplicationManager ist eine Facade-Klasse. [39] Sie stellt selbst wenig Funktionalität zur Verfügung, ist aber dafür zuständig, dem Unity-Frontend eine vereinheitlichte Schnittstelle zum Aufrufen von Funktionalität zur Verfügung zu stellen. Der Quellcode ist im Anhang unter A.2.1 zu finden. Nebst Referenzen zu allen anderen verwendeten Klassen kann sie mittels der Methode `RunPreChecks()` (siehe Zeile 55) evaluieren, ob die Anwendung lauffähig ist. Ausserdem ist in ihr die Methode `StartApplication`, (siehe Zeile 34) implementiert, welche alle benötigten weiteren Komponenten startet und zur Ausführung bringt.

#### Klasse BrowserManager

Wie unter Kapitel 3.4.1 sowie 3.4.2 diskutiert, werden zur Darstellung der Erde sowie zur Aktivierung des virtuellen Display-Treibers 2 Browser-Instanzen benötigt. Die Funktionalität, um diese Browser-Instanzen steuern zu können, ist in der Klasse `BrowserManager` implementiert. Der Quellcode ist im Anhang unter A.2.2 zu finden. Für die Steuerung der Browser-Instanzen haben wir den Selenium Web Driver [40] verwendet. Die beiden Browser-Instanzen sind durch 2 Klassenvariablen

des Typs `IWebDriver` repräsentiert. Diese werden von den verschiedenen Methoden der Klasse angesprochen, um das Verhalten der jeweiligen Browser-Instanz zu kontrollieren.

Die Methode `StartDisplayBrowser()` (siehe Zeile 27) startet den virtuellen Display-Treiber, indem sie eine Browser-Instanz erstellt und zur entsprechenden URL navigiert (vgl. Kap. 3.4.2). Nachdem die Verbindung hergestellt worden ist, wird eine Javascript-Datei geladen. Diese enthält Anweisungen, um die Darstellung des virtuellen Displays zu konfigurieren. Zu guter Letzt werden diese Anweisungen mittels der Methode `ExecuteScriptOnDriver(string script)` (siehe Zeile 58) zur Ausführung gebracht. Diese Methode illustriert beispielhaft, wie beliebiger Javascript-Code auf der Browser-Instanz ausgeführt werden kann.

Die Methode `StartContentBrowser()` (siehe Zeile 77) erstellt die Browser-Instanz, welche die Inhalte effektiv darstellt (vgl. Kap. 3.4.1). Nachdem sie aktiviert wurde, wird sie automatisch auf dem virtuellen Display in den Vollbild-Modus versetzt und eine Willkommens-Animation wird dargestellt.

### Klasse `ProjectionManager`

Dies ist die komplexeste Klasse in der gesamten Anwendung. Der Quellcode der Klasse ist im Anhang unter A.2.3 zu finden. Sie ist dafür verantwortlich, den Input einzulesen, korrekt umzurechnen und darzustellen. Die folgenden Klassenvariablen haben hierfür Bedeutung:

- Darstellung von Texturen
  - `RawImage inputImage`: Enthält die Darstellung, wie sie von ManyCam empfangen wird.
  - `RawImage transformedImage`: Beinhaltet einen Shader, welcher Anweisungen für die Umrechnung enthält und schlussendlich die Darstellung, wie sie an die Projektoren geliefert werden muss.
- Rendern des Outputs durch Kameras
  - `Camera inputCamera`: Rendert auf die Textur, welche dem Anwender den Input anzeigt (vgl. Abb. 18, blau umrandeter Bereich).
  - `Camera projectorCamera1`: Rendert die Aufnahme an das Display, welches am Computer als «Display 2» angeschlossen ist.
  - `Camera projectorCamera2`: Rendert die Aufnahme an das Display, welches am Computer als «Display 3» angeschlossen ist.
  - `Camera projectorCamera1preview`: Rendert auf die Textur, welche dem Anwender den Output an den Projektor 1 anzeigt (vgl. Abb. 18, blau umrandeter Bereich).
  - `Camera projectorCamera2preview`: Rendert auf die Textur, welche dem Anwender den Output an den Projektor 2 anzeigt (vgl. Abb. 18, blau umrandeter Bereich).

Damit die Kameras den richtigen Ausschnitt aufnehmen, müssen die Texturen, auf die der Input respektive der Output gerendert wird, je nach den in den Settings definierten Auflösungen richtig platziert werden. Analog gilt für die Kameras, dass sie sich ebenfalls an der richtigen Position befinden müssen. Unity verwendet ein karthesisches Koordinatensystem, um sämtliche Objekte zu platzieren.

Die Methode `SetupInputView()` (siehe Zeile 44) erledigt diese Aufgaben für die Textur `inputImage` sowie die zugehörige Kamera `inputCamera`. Aus den Einstellungen werden zunächst die benötigte Auflösung sowie das Seitenverhältnis gelesen. Danach wird das Seitenverhältnis auf der Kamera entsprechend gesetzt. Im Anschluss werden Position und Grösse der Textur angepasst, bevor die Position der Kamera aus diesen Informationen ebenfalls gesetzt wird. Um den vertikalen Aufnahmebereich der Kamera (und somit auch den aus dem Seitenverhältnis folgenden horizontalen Aufnahmebereich) zu bestimmen, verwendet Unity den Parameter «Field of View». Dieser Wert wird in Grad angegeben und sagt aus, wie «gross» der Öffnungswinkel der Kamera ist. Um den Öffnungswinkel der Kamera zu errechnen, wird zunächst mittels dem Satz des Pythagoras (Methode `Pythagoras(double a, double b)`, Zeile 116) die fehlende Distanz  $c$  zwischen der Position der Kamera und dem unteren Rand der Textur aus den Werten der Distanz zwischen Kamera und Textur  $b$  sowie der halben Höhe der Textur  $a$  errechnet. Aus den drei gegebenen Seiten errechnet die Methode `CalculateAngle(int a, double c)` (siehe Zeile 121) schlussendlich den halben Öffnungswinkel der Kamera, welcher dann verdoppelt und der Kamera zugewiesen wird.

Die Methode `SetupTransformedView()` (siehe Zeile 73) löst die selbe Aufgabe wie die Methode `SetupInputView()`, allerdings für die Textur `transformedImage` sowie die zugehörigen Kameras. Wie im Kapitel 3.4.5 diskutiert, werden 2 Projektoren benötigt. Die Ausgabe für die beiden Projektoren wird von den Kameras `projectionCamera1` sowie `projectionCamera2` übernommen. Die Kameras mit dem Suffix «preview» zeigen dasselbe wie ihre Pendants ohne das Suffix und rendern auf die Vorschau-Textur des jeweiligen Projektoren (vgl. Abb. 18, blau umrandeter Bereich, «Projector Previews»). Somit sind auch ihre Eigenschaften wie die Position und die Rotation im kartesischen Koordinatensystem identisch zu den Kameras, die ihre Ausgabe auf die Projektoren leiten. Die Positionierung der umrechnenden Textur und der zugehörigen Kameras wird wie folgt erreicht:

- Die umrechnende Textur `transformedImage` wird auf die Grösse  $\text{Projektionsbreite} * \text{Projektionsbreite}$  und an eine Position ohne Kollisionen mit anderen Objekten gesetzt.
- Setzen des Projektionsverhältnisses auf allen Kameras aus den Einstellungen.
- Die Kameras werden über der umrechnenden Textur `transformedImage` positioniert.
  - Die x-Koordinate ergibt sich aus der Position der umrechnenden Textur.
  - Die z-Koordinate ergibt sich aus der Distanz zwischen Kamera und umrechnender Textur.
  - Die y-Koordinate für die Kamera `projectionCamera1` errechnet sich wie folgt:  $y\text{-Koordinate der umrechnenden Textur} + \text{die halbe Höhe der umrechnenden Textur} - \text{die halbe Auflösung des Projektors } y$
  - Die y-Koordinate für die Kamera `projectionCamera2` errechnet sich wie folgt:  $y\text{-Koordinate der umrechnenden Textur} - \text{die halbe Höhe der umrechnenden Textur} + \text{die halbe Auflösung des Projektors } y$
- Der Aufnahmebereich «Field Of View» der Kameras wird analog zur Methode `SetupInputView()` mittels der Methode `Pythagoras(double a, double b)` (siehe Zeile 116) für die fehlende Distanz zwischen Kamera und unterem respektive oberem Rand der umrechnenden Textur und der Methode `CalculateAngle(int a, double c)` (siehe Zeile 121) errechnet.

Die Methode `AdjustProjection(AdjustAction action)` (siehe Zeile 127) implementiert die Funktion des Verschiebens der Kameras im kartesischen Koordinatensystem von Unity. Je nach spezifiziertem Projektor, Aktion sowie der Verschiebedistanz werden die dafür zuständigen Methoden aufgerufen, welche die Verschiebung respektive Spiegelung vornehmen.

Die Methode `SetupDisplays()` (siehe Zeile 222) aktiviert die Displays 2 und 3 (sofern bereits in den Windows-Anzeigeeinstellungen konfiguriert). Sobald die aktiviert sind, liefert Unity die Aufnahme der Kameras `projectionCamera1` sowie `projectionCamera2` an die Displays 2 und 3 aus.

Die Methode `RenderManyCam()` (siehe Zeile 252) sucht aus allen auf dem System installierten Webcams die virtuelle Webcam von «ManyCam» aus und weist deren Textur der Input-Textur `inputImage` und der umrechnenden Textur `transformedImage` zu.

### Klasse SettingsManager

Die Klasse SettingsManager ist zuständig für das Speichern, Validieren und Laden der Einstellungen. Die Methode `SaveSettings()` (Zeile 28) liest die Einstellungen aus den entsprechenden Feldern, validiert sie mittels anderer Methoden und speichert sie schlussendlich in der Registry. Die Methode `LoadSettings()` (Zeile 70) liest die Einstellungen aus der Registry.

## 4 Resultate

In diesem Kapitel werden die Resultate der im Kapitel 3 beschriebenen Implementierungen gezeigt. Die Bewertung aus verschiedenen Perspektiven, insbesondere die Gegenüberstellung zur Aufgabenstellung, wird im Kapitel 5 diskutiert.

### 4.1 Visualisierung mittels Orbitarium

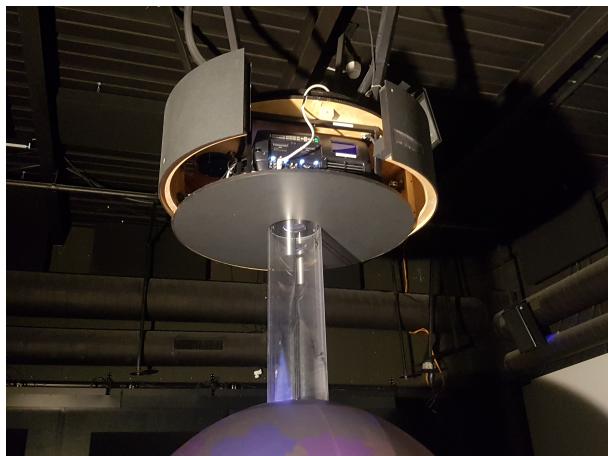
Um das Resultat validieren zu können, wurde die Orbitarium Steuerkonsole im Orbitarium selbst getestet.

#### 4.1.1 Versuchsaufbau

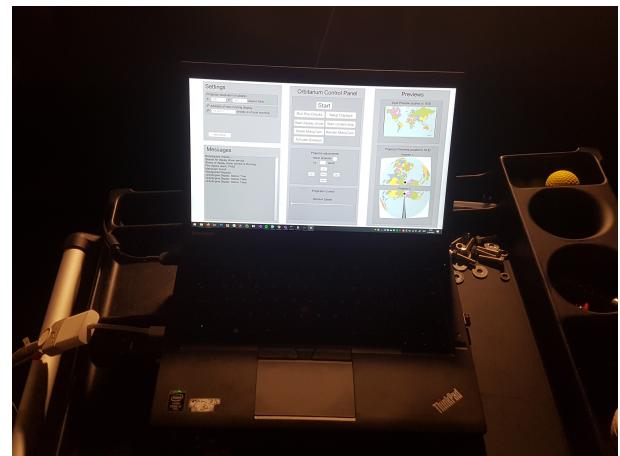
Für das Orbitarium wurde aktuelle, zeitgemäße Hardware evaluiert (siehe Kapitel 3.3). Aufgrund spezieller Umstände war es nicht möglich, die Hardware rechtzeitig zu beschaffen. Deshalb mussten die Resultate in einem reduzierten Setup durchgeführt werden. Der Versuchsaufbau im Orbitarium sieht wie folgt aus:

- Als Plattform für das Orbitarium Orbitarium Control Panel (siehe 3.4.6) kommt ein Notebook zum Einsatz
- Um die Ausgabe zu projizieren, wird einer der bereits vorhandenen Projektoren im Orbitarium genutzt.

Aufgrund der Tatsache, dass das verwendete Notebook nur einen Ausgang für externe Monitore aufweist, wurde nur ein einzelnen Projektor verwendet. Dies genügt jedoch vollends, um einen Eindruck der Lösung zu erhalten. Abbildung 19 veranschaulicht den beschriebenen Aufbau.



(a)



(b)

Abbildung 19: Foto (a) zeigt den zur Validierung eingesetzten Projektor im Orbitarium. Foto (b) zeigt das verwendete Notebook. Links des Notebooks ist der Bild-Ausgang zum Projektor zu sehen.

#### 4.1.2 Versuchsvorbereitungen

Nachdem der Versuchsaufbau vorbereitet war, wurde das Notebook entsprechend konfiguriert. Folgende Konfigurationsmassnahmen wurden getroffen:

- Der am Notebook angeschlossene Projektor wurde in den Windows-Anzeigeeinstellungen als «Display 2» festgelegt.
- Das virtuelle Display wurde hinzugefügt und als «Display 3» in den Windows-Anzeigeeinstellungen konfiguriert.
- Ein Browser wurde mit der Darstellung einer Merkator-Projektion der Erde gestartet und auf das virtuelle Display verschoben.
- Das «Orbitarium Control Panel» wurde gestartet.

#### 4.1.3 Versuchsdurchführung

Nachdem alles vorbereitet wurde, konnte mit der Versuchsdurchführung begonnen werden. Zunächst wurden auf dem Control Panel die Pre-Checks durchgeführt, um die Lauffähigkeit der Applikation sicherzustellen. Danach wurden die internen Verarbeitungsobjekte der Applikation mittels Klick auf «Setup Displays» vorbereitet. Ein Klick auf «Render ManyCam» gefolgt von einem Klick auf «Activate Displays» zeigte schliesslich das umgerechnete Bild auf der Projektionskugel. Mittels der entsprechenden Rotationskomponente wurde die Darstellung in Rotation versetzt. Für eine ausführliche Erklärung des Orbitarium Control Panel sowie der einzelnen Funktionen siehe Kapitel 3.4.6

#### 4.1.4 Versuchsresultate

Die einzelnen Bilder in Abbildung 20 zeigen das projizierte Bild auf dem Globus in den drei relevanten Perspektiven. Wie bereits erwähnt, konnte nur ein Beamer verwendet werden, was man auch anhand der scharfen Kannte am Bildende sehen kann.



(a)



(b)



(c)

Abbildung 20: Die drei relevanten Ansichten auf den Globus (von links (a), frontal (b) und von rechts (c))

# 5 Diskussion / Ausblick

Dieses Kapitel widmet sich der Diskussion und der kritischen Evaluation der erreichten Resultate, welche im vorhergehenden Kapitel 4 aufgezeigt wurden. Die Resultate werden interpretiert und die Frage erörtert, welchen Mehrwert sie bieten. Außerdem werden die erzielten Resultate in den Kontext der Aufgabenstellung gesetzt und es wird erörtert, inwiefern die Ziele erreicht wurden beziehungsweise wo noch Nachholbedarf besteht. Zu guter Letzt wird im Ausblick festgehalten, welches mögliche Szenarien für die Zukunft sind, was kurzfristig noch erledigt werden muss und wie man an diese Arbeit anknüpfen kann.

## 5.1 Interpretation und Mehrwert

Bei der Interpretation der Resultate wird die ursprüngliche Lösung als Referenz genommen. Aus diesem Grund werden nachstehend die zu beachtenden Kriterien nüchtern betrachtet, kategorisch miteinander verglichen und wo nötig, allfällige Anmerkungen dazu gemacht.

- **Output**

- *Auflösung:* Im Vergleich zur ursprünglichen Lösung ist die Auflösung des entwickelten Systems einiges schlechter. Die Kanten sind verschwommen und die Farben wirken blass.
- *Inhalt:* Das Motiv beziehungsweise der Inhalt der Darstellung ist gleich geblieben. Im Vergleich zu früher ist es nun möglich, den Inhalt nach Belieben zu wählen und in Echtzeit zu ändern.
- *Helligkeit:* Im aktuellen Zustand werden die gleichen Beamer verwendet, was bedeutet, dass die Helligkeit und somit die Klarheit des Bildes gleich bleibt.

Wie bereits erwähnt wurde bei der Auflösung eine Verschlechterung gegenüber dem Urzustand festgestellt. Grund dafür ist unzureichendes beziehungsweise ungeeignetes Quellmaterial. In der aktuellen Version arbeiten wir mit einer Auflösung von 1920 \* 1080 Pixel (welche jedoch softwareseitig auf bis zu 3840 \* 2160 Pixel erhöht werden kann, sobald passende Projektoren vorhanden sind), was voraussetzt, dass der Input ebenfalls diese Auflösung besitzen muss. Des Weiteren wird im Input eine Merkatorprojektion erwartet. Da jedoch Merkatorprojektionen von Natur aus ein völlig anderes Seitenverhältnis verglichen mit den Anforderungen aufweisen, müssen entweder an der Auflösung oder am darstellbaren Bereich gewisse Abstriche vorgenommen werden.

- **Performance** Die Performance ist im aktuellen Zustand deutlich schlechter als zuvor. Da der Inhalt in Realtime umgerechnet und gerendert wird, ist der Bedarf an Rechenleistung einiges höher als derjenige des Vorgängersystems. Eine starke Grafikkarte und ausreichend RAM sind daher vonnöten. Eine Anforderung, mit welcher der kleine Office-Laptop, welcher laut Kapitel 4.1.1 für die Darstellung der Resultate verwendet wurde, nicht zurecht kommt. Mehr dazu im Ausblick, Kapitel 5.3.

- Konfiguration und Kontrolle

- *Zentralisierung:* Die umgesetzte Architektur erlaubt es, einen zentralen Punkt zur Steuerung des Systems als Einstiegspunkt zu erhalten. Zuvor ist von einem Steuerungs-PC aus auf den Darstellungs-PC zugegriffen worden, um die Konfiguration und die Darstellung anzupassen. Dies wird nun von einem einzigen PC aus erreicht. Mehr dazu im Kapitel 3.4.6 auf Seite 32.
- *Manipulation der Animation:* Während bisher die Rotationsgeschwindigkeit und die Ausrichtung der Kameras auf die beiden Hälften der Erde Teil der Vorarbeit waren und dementsprechend zur Laufzeit nicht veränderbar gewesen sind, finden sich diese Optionen im «Orbitarium Control Panel» wieder und können in Echtzeit angepasst werden.
- *Wahl des Inputs:* Durch die Darstellung des Inhalts in einem Browser, welcher auf einem virtuellen Display dargestellt wird und von einer Webcam-Software aufgenommen wird, sind die Darstellungsmöglichkeiten unbeschränkt. Der Inhalt kann jederzeit geändert und angepasst werden. Es muss jedoch beachtet werden, dass die Umrechnung von einer Merkatorprojektion ausgeht und deshalb der Input unter Umständen verzerrt wiedergegeben wird.
- *Benutzerfreundlichkeit:* Die Benutzung dieses Tools zur Konfiguration des Systems ist nicht für unerfahrene Benutzer gedacht und nur mit Anleitung zu bedienen. Im Gegensatz dazu ist das frühere Kontrollprogramm durch integrierte Hilfestellung und ausführlicher Dokumentation einiges zugänglicher.

Zusammengefasst wird durch eine bessere Darstellung, weniger verwendeter Hardware, der Echtzeitfähigkeit bei der Umrechnung von Inhalten sowie der flexiblen Wahl des Inputs ein Mehrwert geschaffen. Einbussen werden momentan noch im Bereich der Qualität des Outputs und der Benutzerfreundlichkeit verzeichnet.

## 5.2 Resultat im Vergleich zur Aufgabenstellung

Um einen schlüssigen Vergleich zwischen dem resultierenden System und der Aufgabenstellung ziehen zu können, werden die Kategorien *Hardware* und *Software* einzeln betrachtet. Hierbei werden die umgesetzten Resultate mit der Aufgabenstellung (Anhang A.1 auf Seite 48) und den Zielsetzungen aus Abschnitt 1.3 auf Seite 9 verglichen.

### Hardware

Laut Aufgabenstellung und Zielsetzung sollen Projektoren mit hoher Auflösung und Helligkeit eingebaut werden, ein neuer PC mit geeigneter Leistung soll evaluiert und nach Möglichkeiten beschafft werden und der Globus sollte an einer passenden Aufhängung an der ZHAW in einem geeigneten Raum aufgehängt werden.

Die geforderte Hardware konnte im Zeitfenster dieses Projektes nicht beschafft werden und das Setup musste gemäss der dargelegten Resultate in Kapitel 4 improvisiert und mit einigen Kompromissen umgesetzt werden. Grund dafür sind vor allem administrative Wartezeiten und fehlende Bewilligungen seitens des Geldgebers. Zum aktuellen Zeitpunkt ist die Grafikkarte bestellt und auf dem Weg, die Projektoren stehen zur Verfügung, sind jedoch noch nicht eingebaut und die Aufhängung für den neuen Standort des Orbitariums an der ZHAW ist in Auftrag gegeben worden. Mehr dazu im Ausblick weiter unten.

## **Software**

Der Fokus dieses Projektes lag laut den Vorgaben auf der Entwicklung und Umsetzung einer neuen Architektur für das Gesamtsystem des Orbitariums. Die Recherchen, die Evaluation der Risiken und die Wahl des Vorgehens ist in den Kapiteln 1.1 bis 3.1.4 ausführlich dokumentiert.

Die eigentliche Software für das Projekt wurde komplett erneuert und folgende Punkte, jeweils verglichen mit der Zielsetzung, wurden umgesetzt:

- Mit Hilfe von Unity und der Möglichkeit, auf der Grafikkarte die fordernden Berechnungen auszuführen ist es möglich, die Darstellung auf dem Globus in Echtzeit zu berechnen und zu rendern.
- ManyCam und ein virtueller Bildschirm ermöglichen es, den Inhalt des virtuellen Monitors direkt als Stream an Unity zu übergeben. Somit kann auf dem Globus beliebiges Bildmaterial gezeigt werden. Die einzige Anforderung dabei ist, dass die Quelle die korrekte Auflösung und eine Merkatorprojektion zur Verfügung stellen muss.
- Den *Single Point of Entry* in das System bildet das *Orbitarium Control Panel*. Ausgehend von diesem Tool können die benötigten Einstellungen und externen Tools zentral gestartet und konfiguriert werden.

Abschliessend betrachtet wurde das Ziel bezüglich Architektur und Software laut den Vorgaben erreicht. Seitens Hardware sind einige der Ziele in der gegebenen Zeit nicht erreicht worden.

## **5.3 Ausblick**

Es wurde gezeigt, dass dieses Projekt einiges an Potenzial hat. Mit den künftigen Verbesserungen im Bereich der Projektoren werden solche Darstellungen immer realistischer und erschwinglicher. Hier diskutieren wir zukünftige Entwicklungsmöglichkeiten für das Orbitarium:

### **Kurzfristig**

Die nächsten Schritte widmen sich der Umsetzung offener Zielsetzungen. Als erstes ist die Beschaffung der Hardware abzuschliessen. Wie bereits erwähnt sind sowohl Grafikkarte als auch Aufhängung bereits bestellt. Die Projektoren stehen ebenfalls zur Verfügung und müssen noch eingebaut werden. Ein passender Raum an der ZHAW ist ebenfalls gefunden. So sollte es in kurzer Zukunft möglich sein, das Orbitarium an die ZHAW zu transportieren und dort aufzustellen. Sobald das Orbitarium steht können die nötigen Feinjustierungen der Darstellung in Zusammenarbeit mit der neuen Hardware vorgenommen werden.

## **Mittelfristig**

Interessant sind vor allem die mittelfristigen Möglichkeiten und Ansatzpunkte. Aufgrund von defekter Hardware wurde in diesem Projekt stets mit einem Beamer gearbeitet. Sobald jedoch zwei oder mehr Beamer eine Darstellung projizieren, stösst man auf das Problem der Kantenübergänge. Eine Arbeit könnte sich nun beispielsweise damit befassen, einen Algorithmus zu entwickeln, diese Kantenübergänge verschwinden.

Als weitere Möglichkeit wäre die Entwicklung weiterer Shader plausibel. Ein Ansatzpunkt wäre beispielsweise die automatische Umrechnung eines beliebigen Bildes in die Merkatorprojektion, damit diese wiederum als Input für das Orbitarium dienen könnte.

Zudem ist mittelfristig die Entwicklung von Animationen anzusetzen. Leider war es uns aufgrund der zeitlichen Begebenheiten nicht möglich, eine eigene Animation für das Orbitarium zu entwickeln.

## **Langfristig**

Weiter als ein paar Monate in die Zukunft zu blicken ist in einer schnelllebigen Branche wie der Informatik oft sehr schwierig und die Aussagen meist ungenau und zum künftigen Zeitpunkt veraltet. Es wird jedoch die Möglichkeit in Betracht gezogen, das Orbitarium an das Verkehrshaus weiter zu verkaufen. Daraus könnte ein weiteres Projekt entstehen mit spezifischen Vorgaben und individualisierten Anforderungen.

## 6 Verzeichnisse

### 6.1 Literaturverzeichnis

## Literatur

- [1] Technorama. *Technorama - Über uns.* 2020. URL: <https://www.technorama.ch/de/ueber-uns> (besucht am 26.05.2020).
- [2] ARC Science Simulations. *ARC Science Simulations Inc. – OmniGlobe Spherical Displays.* 2020. URL: <https://arcscience.com/> (besucht am 11.05.2020).
- [3] ARC Science Simulations. *OmniGlobe Technology.* 2020. URL: <https://arcscience.com/omniglobe-technology/> (besucht am 11.05.2020).
- [4] ARC Science Simulations. *OmniGlobe Software.* 2020. URL: <https://arcscience.com/omniglobe-software/> (besucht am 11.05.2020).
- [5] National Oceanic und Atmospheric Administration. *What is SOS?* 2020. URL: [https://sos.noaa.gov/What\\_is\\_SOS/](https://sos.noaa.gov/What_is_SOS/) (besucht am 11.05.2020).
- [6] National Oceanic und Atmospheric Administration. *Science On a Sphere Wiring Diagram.* 2020. URL: [https://sos.noaa.gov/\\_media/cms/docs/SOS\\_Wiring\\_Diagram.pdf](https://sos.noaa.gov/_media/cms/docs/SOS_Wiring_Diagram.pdf) (besucht am 11.05.2020).
- [7] National Oceanic und Atmospheric Administration. *SOS Product Suite.* 2020. URL: <https://sos.noaa.gov/support/sos-product-suite/#visual-playlist-editor> (besucht am 11.05.2020).
- [8] National Oceanic und Atmospheric Administration. *SOS Datasets.* 2020. URL: <https://sos.noaa.gov/Datasets/> (besucht am 11.05.2020).
- [9] National Oceanic und Atmospheric Administration. *Installation Gallery.* 2020. URL: <https://sos.noaa.gov/InstallationGallery/> (besucht am 11.05.2020).
- [10] Unity. *Unity - Multiplatform.* 2020. URL: <https://unity.com/features/multiplatform> (besucht am 29.05.2020).
- [11] Heiskanen V., Moritz H. *Physical Geodesy.* Springer, 2005.
- [12] Wikipedia. *List of Map Projections.* 2020. URL: [https://en.wikipedia.org/wiki/List\\_of\\_map\\_projections](https://en.wikipedia.org/wiki/List_of_map_projections) (besucht am 29.05.2020).
- [13] Borradaile, Graham J. *Statistics of Earth Science Data.* Springer, 2003.
- [14] Google Maps und Earth Help Forum. *Why Google Maps uses Mercator Projection.* 2020. URL: [https://support.google.com/maps/forum/AAAAQuUrST8A2ygEJ5eG-o/?hl=en&msgid=KbZr\\_B0h2hkJ&gpf=d/msg/maps/A2ygEJ5eG-o/KbZr\\_B0h2hkJ](https://support.google.com/maps/forum/AAAAQuUrST8A2ygEJ5eG-o/?hl=en&msgid=KbZr_B0h2hkJ&gpf=d/msg/maps/A2ygEJ5eG-o/KbZr_B0h2hkJ) (besucht am 29.05.2020).
- [15] Luca Fontana. *DLP meets LCD: Wie funktionieren Beamer?* 2018. URL: <https://www.digitec.ch/de/page/dlp-meets-lcd-wie-funktionieren-beamer-7126> (besucht am 31.05.2020).
- [16] Markus Ries. *LED-Projektoren.* 2016. URL: <https://www.professional-system.de/basics/laser-beamer-mit-laser-in-die-zukunft/#laserbeamer2> (besucht am 01.06.2020).
- [17] Markus Ries. *LED-Projektoren.* 2016. URL: <https://www.professional-system.de/basics/laser-beamer-mit-laser-in-die-zukunft/#laserbeamer4> (besucht am 01.06.2020).

- [18] Markus Ries. *LED-Projektoren*. 2016. URL: <https://www.professional-system.de/basics/laser-beamer-mit-laser-in-die-zukunft/#laserbeamer5> (besucht am 01.06.2020).
- [19] Zumtobel Lighting GmbH. *Licht-Handbuch für den Praktiker*. 2018. URL: <https://www.zumtobel.com/PDB/teaser/DE/Lichthandbuch.pdf> (besucht am 01.06.2020).
- [20] Beleuchtungdirekt GmbH. *Lux-Vergleichswerte*. 2020. URL: <https://www.beleuchtungdirekt.at/blog/lumen-lux-unterschied> (besucht am 01.06.2020).
- [21] Atelier Rieter GmbH. *Neu: Wie hell sollte ein Beamer sein?* 2020. URL: [https://www.atelier-rieter.de/Beamer\\_Info\\_Lichtstaerke.htm](https://www.atelier-rieter.de/Beamer_Info_Lichtstaerke.htm) (besucht am 01.06.2020).
- [22] Grossleinwand GmbH. *Ratio, Distanz, Bildbreite*. 2020. URL: <https://www.grossleinwand.ch/glw/beratung-offerte/ratio-distanz-bildbreite.html> (besucht am 01.06.2020).
- [23] Grossleinwand GmbH. *Lens-Shift*. 2020. URL: <https://www.grossleinwand.ch/glw/fachbegriffe/lens-shift/> (besucht am 01.06.2020).
- [24] TeamViewer. *Das ist TeamViewer*. 2020. URL: <https://www.teamviewer.com/de/produkte/teamviewer/> (besucht am 26.05.2020).
- [25] Visicom Media Inc. *ManyCam - Live Video made better*. 2020. URL: <https://manycam.com/> (besucht am 02.06.2020).
- [26] ProjectorCentral. *projectiondesign FL32 1080p Projector*. 2020. URL: [https://www.projectorcentral.com/projectiondesign-FL32\\_1080p.htm](https://www.projectorcentral.com/projectiondesign-FL32_1080p.htm) (besucht am 31.05.2020).
- [27] Lang Baranday. *Leidenschaft für Technik*. 2020. URL: <https://www.lang-baranday.ch/> (besucht am 02.06.2020).
- [28] Barco GmbH. *FL40-4K*. 2020. URL: <https://www.barco.com/de/product/fl40-4k> (besucht am 02.06.2020).
- [29] NVIDIA Corporation. *RTX 2080 Ti*. 2020. URL: <https://www.nvidia.com/de-de/geforce/graphics-cards/rtx-2080-ti/#specs> (besucht am 02.06.2020).
- [30] Dell Corporation. *Precision Tower 5810*. 2020. URL: <https://www.dell.com/support/home/de-ch/product-support/product/precision-t5810-workstation/docs> (besucht am 02.06.2020).
- [31] Flightradar24 AB. *Flightradar 24: Live Flight Tracker Map*. 2020. URL: <https://www.flightradar24.com/> (besucht am 02.06.2020).
- [32] S.E. Windyty. *Windy: Wind map & weather forecast*. 2020. URL: <https://www.windy.com> (besucht am 02.06.2020).
- [33] Google LLC. *Google Maps*. 2020. URL: <https://www.google.com/maps> (besucht am 02.06.2020).
- [34] Google LLC. *Keyhole Markup Language*. 2020. URL: <https://developers.google.com/kml> (besucht am 02.06.2020).
- [35] The OpenSky Network. *Open Air Traffic Data for Research*. 2020. URL: <https://opensky-network.org/> (besucht am 02.06.2020).
- [36] Google LLC. *KML Documentation Introduction*. 2020. URL: <https://developers.google.com/kml/documentation> (besucht am 02.06.2020).
- [37] datronicssoft UG. *Windows network display monitor software*. 2020. URL: <https://spacedesk.net/> (besucht am 02.06.2020).

- [38] Unity Inc. *Scripting API: WebCamTexture*. 2020. URL: <https://docs.unity3d.com/ScriptReference/WebCamTexture.html> (besucht am 02.06.2020).
- [39] Craig Larman. *UML2 und Patterns angewendet*. 1. Aufl. Seite 471. mitp-Verlag, 2005. ISBN: 978-3-8266-1453-8.
- [40] Software Freedom Conservancy. *Selenium WebDriver*. 2020. URL: <https://www.selenium.dev/documentation/en/webdriver/> (besucht am 12.06.2020).

## 6.2 Abbildungsverzeichnis

### Abbildungsverzeichnis

1	Die Projektionsgeometrie des ARC Science OmniGlobe . . . . .	8
2	Eine «Science On a Sphere»-Installation . . . . .	9
3	Das Orbitarium . . . . .	11
4	Rektangularprojektion mit tissotscher Indikatrix. . . . .	13
5	Merkatorprojektion mit tissotscher Indikatrix. . . . .	13
6	Azimutalprojektion der Erde mit eingezeichneter Tissotscher Indikatrix. . . . .	14
7	Aufhängungspunkt Metallplatte . . . . .	19
8	Lichtfluss Orbitariums . . . . .	19
9	Projektionsgeometrie Orbitariums . . . . .	20
10	Aufsicht auf den Projektionsspiegel . . . . .	21
11	Risikoanalyse der Arbeit . . . . .	21
12	Die Vision des Gesamtsystems . . . . .	24
13	Ansicht des Projektorkäfigs . . . . .	27
14	Google Maps mit eigenem KML-Overlay . . . . .	28
15	Windows-Anzeigeeinstellungen . . . . .	30
16	Einstellungen virtueller Monitor . . . . .	30
17	Sehr rudimentäre Darstellung der Canvas und der beiden Kameras. . . . .	32
18	Die Orbitarium Control Panel . . . . .	33
19	Der zur Validierung eingesetzte Projektor mit Notebook. . . . .	38
20	Verschiedene Ansichten auf die Globusprojektion . . . . .	39

# A Anhang

## A.1 Aufgabenstellung

Zürcher Hochschule  
für Angewandte Wissenschaften



### Praktische Bachelorarbeit

Dozent:	Karl Rege	Studiengang:	IT
		:	2020
Industriepartner:	Jahr:		
Studierende:	Remo Preuss, preusrem (IT) Nicolas Philipp Schaller, schalnic (IT)	Ausgabe:	10.02.20
		Abgabe:	05.06.20

### ZHAW Orbitarium

Die ZHAW übernimmt das Orbitarium des Technoramas. Dabei sollen die HW/SW dem aktuellen technischen Stand angepasst werden. Bei der HW geht es primär um den Einsatz von helleren Projektoren mit einer grösseren Auflösung, was keinen grossen Aufwand darstellt. Die SW hingegen muss stark erneuert werden. Die Details dieser Erweiterungen werden noch mit dem Entwickler des Orbitariums und der NOAA festgelegt.

#### Aufgabenstellung

- Erstellen Sie einen Projektplan
- Bestimmen Sie die grössten Risiken des Projekts
- Recherchieren Sie den aktuellen Stand der Technik
- Entwickeln Sie eine Architektur und die Anwendung
- Testen und verifizieren Sie die Resultate
- Betrachten Sie die erzielten Resultate kritisch und überlegen Sie sich mögliche Erweiterungen
- Eine technische Dokumentation (= BA Bericht) und ev. ein einfaches Benutzerhandbuch (im Anhang) soll erstellt werden

## A.2 Code Listings

Die Imports sind jeweils ausgelassen.

### A.2.1 Application.cs

```
13 public class ApplicationManager : MonoBehaviour
14 {
15     private LogManager log;
16     private BrowserManager browserManager;
17     private ProjectionManager projectionManager;
18     private SettingsManager settings;
19
20     void Start()
21     {
22         log = new LogManager();
23         browserManager = new BrowserManager(log);
24         projectionManager = new ProjectionManager(log);
25         settings = new SettingsManager(log);
26         settings.LoadSettings();
27     }
28
29     void Update()
30     {
31
32     }
33
34     public void StartApplication()
35     {
36         log.LogWrite("Starting Application");
37         try
38         {
39             RunPreChecks();
40             projectionManager.SetupInputView();
41             browserManager.StartDisplayBrowser();
42             System.Threading.Thread.Sleep(5000);
43             browserManager.StartContentBrowser();
44             projectionManager.StartManyCam();
45             projectionManager.RenderManyCam();
46             projectionManager.SetupDisplays();
47
48         } catch (Exception e)
49         {
50             log.LogWrite(e.Message);
51             log.LogWrite("Application failed to start.");
52         }
53     }
54
55     public void RunPreChecks()
56     {
57         log.LogWrite("Running pre checks...");
58         try
59         {
60             settings.CheckSettings();
61             browserManager.CheckForDisplayDriver();
62         } catch (Exception e)
```

```

63
64         {
65             throw new Exception(e.Message + "\nPre checks result: FAIL");
66         }
67         log.LogWrite("Pre checks result: PASS");
68     }
69
70     public void StartDisplayBrowser()
71     {
72         browserManager.StartDisplayBrowser();
73     }
74
75     public void StartContentBrowser()
76     {
77         browserManager.StartContentBrowser();
78     }
79
80     public void StartDisplayRecorder()
81     {
82         projectionManager.StartManyCam();
83     }
84
85     public void RenderDisplayRecorder()
86     {
87         projectionManager.RenderManyCam();
88     }
89
90     public void SetupInternalViews()
91     {
92         projectionManager.SetupInputView();
93         projectionManager.SetupTransformedView();
94     }
95
96     public void SaveSettings()
97     {
98         settings.SaveSettings();
99     }
100
101    public void AdjustProjector(string action)
102    {
103        if (action.Equals("up"))
104        {
105            projectionManager.AdjustProjection(AdjustAction.UP);
106        }
107        else if (action.Equals("down"))
108        {
109            projectionManager.AdjustProjection(AdjustAction.DOWN);
110        }
111        else if (action.Equals("right"))
112        {
113            projectionManager.AdjustProjection(AdjustAction.RIGHT);
114        }
115        else if (action.Equals("left"))
116        {
117            projectionManager.AdjustProjection(AdjustAction.LEFT);
118        }
119        else if (action.Equals("mirror"))

```

```

120     {
121         projectionManager.AdjustProjection(AdjustAction.MIRROR);
122     }
123 }
124
125 public void CheckDisplays()
126 {
127     projectionManager.SetupDisplays();
128 }
129 }
```

### A.2.2 BrowserManager.cs

```

16 public class BrowserManager
17 {
18     private IWebDriver displayBrowser;
19     private IWebDriver contentBrowser;
20     private LogManager log;
21
22     public BrowserManager(LogManager log)
23     {
24         this.log = log;
25     }
26
27     public void StartDisplayBrowser()
28     {
29         log.LogWrite("Starting display driver...");
30         //Configure chrome options (hide indication that browser is controlled
31         by selenium).
32         ChromeOptions options = new ChromeOptions();
33         options.AddExcludedArgument("enable-automation");
34         options.AddAdditionalCapability("useAutomationExtension", false);
35         //navigate to url
36         displayBrowser = new ChromeDriver(options);
37         displayBrowser.Navigate().GoToUrl("http://viewer.spacedesk.net/");
38         //wait for element "server" to be present
39         WebDriverWait wait = new WebDriverWait(displayBrowser, TimeSpan.
FromSeconds(10));
40         wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.
ElementToBeClickable(By.CssSelector("#server")));
41         //clear ip field
42         displayBrowser.FindElement(By.Id("server")).Clear();
43         //write ip address address of network interface
44         displayBrowser.FindElement(By.Id("server")).SendKeys(PlayerPrefs.
GetString("ip"));
45         //connect
46         displayBrowser.FindElement(By.Id("buttonLogin")).Click();
47         log.LogWrite("Display driver started successfully.");
48         displayBrowser.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds
(10);
49         wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.
ElementToBeClickable(By.CssSelector("#toggleMenu")));
50         //getting js for configuration of display driver
51         var basePath = Directory.GetCurrentDirectory();
52         var scriptPath = basePath + @"\Assets\Scripts\js\displaydriver_config.js
";
53         var script = GetScript(scriptPath);
```

```

53     script += "$( document ).ready(function() {setUpDisplay(" + PlayerPrefs.
54 GetInt("resx") + "," + PlayerPrefs.GetInt("resy") + ");});";
55     ExecuteScriptOnDriver(script);
56     log.LogWrite("Display driver configured successfully.");
57 }
58
59 private void ExecuteScriptOnDriver(string script)
60 {
61     IJavaScriptExecutor js = (IJavaScriptExecutor)displayBrowser;
62     js.ExecuteScript(script);
63 }
64
65 private string GetScript(string scriptPath)
66 {
67     string line;
68     string script = "";
69     StreamReader file = new System.IO.StreamReader(scriptPath);
70     while ((line = file.ReadLine()) != null)
71     {
72         script += line;
73     }
74     file.Close();
75     return script;
76 }
77
78 public void StartContentBrowser()
79 {
80     ChromeOptions options = new ChromeOptions();
81     options.AddExcludedArgument("enable-automation");
82     options.AddAdditionalCapability("useAutomationExtension", false);
83     contentBrowser = new ChromeDriver(options);
84     contentBrowser.Navigate().GoToUrl("http://localhost/orbitarium.ba/
welcome");
85
86     var wholeWidth = System.Windows.Forms.Screen.PrimaryScreen.Bounds.Width;
87     var xPos = wholeWidth - PlayerPrefs.GetInt("resy");
88     var posPoint = new Point(xPos, 0);
89     contentBrowser.Manage().Window.Position = posPoint;
90     contentBrowser.Manage().Window.FullScreen();
91 }
92
93 public void CheckForDisplayDriver()
94 {
95     log.LogWrite("Search for display driver service");
96     //check if display driver is installed and running
97     ServiceController sc = null;
98     try
99     {
100         sc = new ServiceController("spacedeskService");
101     }
102     catch (Win32Exception)
103     {
104         var ex = new InvalidOperationException("Display driver not found.
Please install spacedesk windows DRIVER");
105         throw ex;
106     }

```

```

107     log.LogWrite("Status of display driver service is:" + sc.Status.ToString
108    ()));
109     if (!sc.Status.Equals(ServiceControllerStatus.Running))
110     {
111         var ex = new InvalidOperationException("Display driver error: The
112         spacedesk display driver is not running.\nPlease start the service.");
113         throw ex;
114     }
115 }
```

### A.2.3 ProjectionManager.cs

```

10 public enum AdjustAction
11 {
12     UP,
13     DOWN,
14     RIGHT,
15     LEFT,
16     MIRROR
17 }
18
19 public class ProjectionManager
20 {
21     private LogManager log;
22     private Process manyCamProc;
23     private RawImage inputImage;
24     private RawImage transformedImage;
25     private Camera inputCamera;
26     private Camera projectorCamera1;
27     private Camera projectorCamera2;
28     private Camera projectorCamera1preview;
29     private Camera projectorCamera2preview;
30
31     public ProjectionManager(LogManager log)
32     {
33         this.log = log;
34         inputImage = GameObject.Find("inputImage").GetComponent<RawImage>();
35         transformedImage = GameObject.Find("transformedImage").GetComponent<
36         RawImage>();
37         inputCamera = GameObject.Find("inputCamera").GetComponent<Camera>();
38         projectorCamera1 = GameObject.Find("projector1").GetComponent<Camera>();
39         projectorCamera2 = GameObject.Find("projector2").GetComponent<Camera>();
40         projectorCamera1preview = GameObject.Find("projectorpreview1") .
41             GetComponent<Camera>();
42         projectorCamera2preview = GameObject.Find("projectorpreview2") .
43             GetComponent<Camera>();
44
45     public void SetupInputView()
46     {
47         var resX = PlayerPrefs.GetInt("resX");
48         var resY = PlayerPrefs.GetInt("resY");
49         var aspectX = Convert.ToSingle(PlayerPrefs.GetInt("aspectX"));
50         var aspectY = Convert.ToSingle(PlayerPrefs.GetInt("aspectY"));
```

```

50     //setting aspect ratio
51     inputCamera.aspect = aspectX / aspectY;
52     var inputImagePosition = new Vector3(4000, 0, -1000);
53     var inputImageSize = new Vector2(resX, resY);
54     //setting image position
55     inputImage.rectTransform.position = inputImagePosition;
56     //setting image size
57     inputImage.rectTransform.sizeDelta = inputImageSize;
58     //setting camera position
59     var cameraPosition = inputImagePosition;
60     cameraPosition.z -= 1000;
61     inputCamera.transform.position = cameraPosition;
62     //calculate input camera distance
63     var camPosz = inputCamera.transform.position.z;
64     var imgPosz = inputImage.transform.position.z;
65     //calculate and set view angle of camera
66     var b = Math.Abs(camPosz - imgPosz);
67     var a = resY / 2;
68     var c = Pythagoras(a, b);
69     var angle = Convert.ToSingle(CalculateAngle(a, c) * 2);
70     inputCamera.fieldOfView = angle;
71 }
72
73 public void SetupTransformedView()
74 {
75     //setting position and size of projection image
76     var resX = PlayerPrefs.GetInt("resX");
77     var resY = PlayerPrefs.GetInt("resY");
78     var transformedImagePosition = new Vector3(4000, 4000, 0);
79     var transformedImageSize = new Vector2(resX, resY);
80     transformedImage.rectTransform.position = transformedImagePosition;
81     transformedImage.rectTransform.sizeDelta = transformedImageSize;
82
83     //set aspect for projection cameras
84     var aspectX = Convert.ToSingle(PlayerPrefs.GetInt("aspectX"));
85     var aspectY = Convert.ToSingle(PlayerPrefs.GetInt("aspectY"));
86     var aspect = aspectX / aspectY;
87     projectorCamera1.aspect = aspect;
88     projectorCamera2.aspect = aspect;
89     projectorCamera1preview.aspect = aspect;
90     projectorCamera2preview.aspect = aspect;
91
92     //setting position of projection cameras
93     var camDistance = 1000;
94     var camerasX = transformedImage.rectTransform.position.x;
95     var camerasZ = transformedImage.rectTransform.position.z - camDistance;
96     var cam1y = transformedImage.rectTransform.position.y + transformedImage
97 .rectTransform.sizeDelta.y / 2 - resY / 2;
98     var cam2y = transformedImage.rectTransform.position.y - transformedImage
99 .rectTransform.sizeDelta.y / 2 + resY / 2;
100    var cam1position = new Vector3(camerasX, cam1y, camerasZ);
101    var cam2position = new Vector3(camerasX, cam2y, camerasZ);
102    projectorCamera1.transform.position = cam1position;
103    projectorCamera2.transform.position = cam2position;
104    projectorCamera1preview.transform.position = cam1position;
105    projectorCamera2preview.transform.position = cam2position;

```

```

105     //calculate and set required field of view for camera
106     var b = camDistance;
107     var a = resY / 2;
108     var c = Pythagoras(a, b);
109     var angle = Convert.ToSingle(CalculateAngle(a, c) * 2);
110     projectorCamera1.fieldOfView = angle;
111     projectorCamera2.fieldOfView = angle;
112     projectorCamera1preview.fieldOfView = angle;
113     projectorCamera2preview.fieldOfView = angle;
114 }
115
116 private double Pythagoras(double a, double b)
117 {
118     return Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 2));
119 }
120
121 private double CalculateAngle(int a, double c)
122 {
123     var angle = Math.Asin(a / c) * (180 / Math.PI);
124     return angle;
125 }
126
127 public void AdjustProjection(AdjustAction action)
128 {
129     var projectorString = GameObject.Find("forProjector").GetComponent<
InputField>().text;
130     Camera projector;
131     Camera projectorPreview;
132     if (projectorString.Equals("1"))
133     {
134         projector = projectorCamera1;
135         projectorPreview = projectorCamera1preview;
136     }
137     else
138     {
139         projector = projectorCamera2;
140         projectorPreview = projectorCamera2preview;
141     }
142
143     if(action != AdjustAction.MIRROR)
144     {
145         var pixels = Int32.Parse(GameObject.Find("byPixels").GetComponent<
InputField>().text);
146         if(action == AdjustAction.UP)
147         {
148             MoveProjectorUp(projector, pixels);
149             MoveProjectorUp(projectorPreview, pixels);
150         } else if (action == AdjustAction.DOWN)
151         {
152             MoveProjectorDown(projector, pixels);
153             MoveProjectorDown(projectorPreview, pixels);
154         }
155         else if (action == AdjustAction.RIGHT)
156         {
157             MoveProjectorRight(projector, pixels);
158             MoveProjectorRight(projectorPreview, pixels);
159         }
}

```

```

160         else if (action == AdjustAction.LEFT)
161     {
162         MoveProjectorLeft(projector, pixels);
163         MoveProjectorLeft(projectorPreview, pixels);
164     }
165 } else
166 {
167     MirrorProjector(projector);
168     MirrorProjector(projectorPreview);
169 }
170 }
171
172 private void MoveProjectorUp(Camera cam, int pixels)
173 {
174     log.LogWrite("moving up by " + pixels);
175     var position = cam.transform.position;
176     var newPosition = new Vector3(position.x, position.y + pixels, position.
z);
177     cam.transform.position = newPosition;
178 }
179
180 private void MoveProjectorDown(Camera cam, int pixels)
181 {
182     log.LogWrite("moving down by " + pixels);
183     var position = cam.transform.position;
184     var newPosition = new Vector3(position.x, position.y - pixels, position.
z);
185     cam.transform.position = newPosition;
186 }
187
188 private void MoveProjectorRight(Camera cam, int pixels)
189 {
190     log.LogWrite("moving right by " + pixels);
191     var position = cam.transform.position;
192     var newPosition = new Vector3(position.x + pixels, position.y, position.
z);
193     cam.transform.position = newPosition;
194 }
195
196 private void MoveProjectorLeft(Camera cam, int pixels)
197 {
198     log.LogWrite("moving left by " + pixels);
199     var position = cam.transform.position;
200     var newPosition = new Vector3(position.x - pixels, position.y, position.
z);
201     cam.transform.position = newPosition;
202 }
203
204
205 private void MirrorProjector(Camera cam)
206 {
207     log.LogWrite("rotate by 180");
208     var rotation = cam.transform.rotation;
209     float newZ;
210     if (rotation.z == 180.0f)
211     {
212         newZ = 0.0f;

```

```

213     }
214     else
215     {
216         newZ = 180.0f;
217     }
218     var newRotation = new Quaternion(rotation.x, rotation.y, newZ, rotation.
w);
219     cam.transform.rotation = newRotation;
220 }
221
222 public void SetupDisplays()
223 {
224     log.LogWrite("Recognized Displays:");
225     var displays = Display.displays;
226     var resX = PlayerPrefs.GetInt("resX");
227     var resY = PlayerPrefs.GetInt("resY");
228     //Activate Display 2 and 3 for projector 1 and 2
229     for (int i = 0; i < displays.Length; i++)
230     {
231         log.LogWrite(displays[i].ToString() + ", Status: " + displays[i].
active);
232         if(i == 1 || i == 2)
233         {
234             displays[i].Activate(resX, resY, 60);
235         }
236     }
237 }
238
239 public void StartManyCam()
240 {
241     log.LogWrite("Starting ManyCam... ");
242     manyCamProc = new Process();
243     var path = @"C:\Program Files (x86)\ManyCam\ManyCam.exe";
244     manyCamProc.StartInfo.UseShellExecute = false;
245     manyCamProc.StartInfo.FileName = path;
246     manyCamProc.StartInfo.CreateNoWindow = false;
247     manyCamProc.Start();
248     manyCamProc.WaitForInputIdle();
249     log.LogWrite("ManyCam successfully started. Please configure it.");
250 }
251
252 public void RenderManyCam()
253 {
254     WebCamDevice[] devices = WebCamTexture.devices;
255     int manyCamId = -1;
256     //search for manycam
257     for (int i = 0; i < devices.Length; i++)
258     {
259         if (devices[i].name.ToLower().Contains("manycam"))
260         {
261             manyCamId = i;
262         }
263     }
264     if (manyCamId == -1)
265     {
266         throw new Exception("ManyCam not found! Please install ManyCam");
267     }
}

```

```

268     log.LogWrite("ManyCam found!");
269     //getting cam feed of manycam
270     WebCamTexture tex = new WebCamTexture(devices[manyCamId].name);
271     inputImage.texture = tex;
272     transformedImage.texture = tex;
273     tex.Play();
274 }
275 }
```

#### A.2.4 SettingsManager.cs

```

10 public class SettingsManager
11 {
12     private readonly LogManager log;
13     private readonly IDictionary<string, string> requiredSettings = new
14     Dictionary<string, string>()
15     {
16         { "resX", "resolution X" },
17         { "resY", "resolution Y" },
18         { "ip", "ip address" }
19     };
20
21     private readonly int MAX_RES = 8000;
22     private readonly int MIN_RES = 100;
23
24     public SettingsManager(LogManager log)
25     {
26         this.log = log;
27     }
28
29     public void SaveSettings()
30     {
31         log.LogWrite("Saving settings...");
32         try
33         {
34             int resX = GetResolution(GameObject.Find("resX").GetComponent<
35             InputField>().text);
36             int resY = GetResolution(GameObject.Find("resY").GetComponent<
37             InputField>().text);
38             string ip = ValidateIP(GameObject.Find("ip").GetComponent<InputField
39             >().text);
40             var aspects = CalculateAspectRatio(resX, resY);
41             var aspectX = aspects.Item1;
42             var aspectY = aspects.Item2;
43             PlayerPrefs.SetInt("resX", resX);
44             PlayerPrefs.SetInt("resY", resY);
45             PlayerPrefs.SetInt("aspectX", aspectX);
46             PlayerPrefs.SetInt("aspectY", aspectY);
47             PlayerPrefs.SetString("ip", ip);
48             PlayerPrefs.Save();
49
50         } catch (Exception e)
51         {
52             log.LogWrite(e.Message);
53             log.ShowSaveSettingsInformation(e.Message, true);
54             log.LogWrite("Error. Settings not saved.");
55             return;
56         }
57     }
58 }
```

```

52     }
53     log.ShowSaveSettingsInformation("Settings successfully saved!", false);
54     log.LogWrite("Settings successfully saved!");
55 }
56
57 private string ValidateIP(string address)
58 {
59     IPAddress ip;
60     bool validIP = IPAddress.TryParse(address, out ip);
61     if (validIP)
62     {
63         return address;
64     } else
65     {
66         throw new FormatException("Entered IP Address is invalid. Please
specify a valid IP-Address in settings.");
67     }
68 }
69
70 public void LoadSettings()
71 {
72     int resx = 0;
73     int resy = 0;
74     if (PlayerPrefs.HasKey("resX"))
75     {
76         resx = PlayerPrefs.GetInt("resX");
77         GameObject.Find("resX").GetComponent<InputField>().text = resx.
ToString();
78     }
79     if (PlayerPrefs.HasKey("resY"))
80     {
81         resy = PlayerPrefs.GetInt("resY");
82         GameObject.Find("resY").GetComponent<InputField>().text = resy.
ToString();
83     }
84     if (PlayerPrefs.HasKey("ip"))
85     {
86         GameObject.Find("ip").GetComponent<InputField>().text = PlayerPrefs.
GetString("ip").ToString();
87     }
88     CalculateAspectRatio(resx, resy);
89 }
90
91 public void CheckSettings()
92 {
93     foreach (KeyValuePair<string, string> entry in requiredSettings)
94     {
95         if (!PlayerPrefs.HasKey(entry.Key))
96         {
97             throw new ArgumentNullException("Required setting '" + entry.
Value + "' missing.\nPlease enter it in 'Settings'.");
98         }
99     }
100 }
101
102 private int GetResolution(string resString)
103 {

```

```

104     int res;
105     bool parseSuccessful = Int32.TryParse(resString, out res);
106     if (!parseSuccessful)
107     {
108         var ex = new FormatException("Could not save the resolution settings
109         .\nPlease specify numbers in the section 'Projector resolution'");
110         throw ex;
111     }
112     if (IsValidResolution(res))
113     {
114         return res;
115     } else
116     {
117         return -1;
118     }
119 }
120 private bool IsValidResolution(int res)
121 {
122     if(res > MAX_RES || res < MIN_RES)
123     {
124         var ex = new ArgumentOutOfRangeException("Specified resolution '" +
125             res + "' is out of range.\nPlease specify a resolution between 100 and 8000
126             pixels.");
127         throw ex;
128     }
129     return true;
130 }
131 private (int, int) CalculateAspectRatio(int resx, int resy)
132 {
133     var ggt = calcGgt(resx, resy);
134     var aspectx = resx / ggt;
135     var aspecty = resy / ggt;
136     if (aspectx < 2 * aspecty)
137     {
138         log.ShowAspectRatio(aspectx, aspecty, true);
139     }
140     else
141     {
142         log.ShowAspectRatio(aspectx, aspecty, false);
143         log.LogWrite("Warning: Detected anomal aspect ratio " + aspectx + ":" +
144             aspecty);
145     }
146     return (aspectx, aspecty);
147 }
148 private int calcGgt(int res1, int res2)
149 {
150     int number1 = res1;
151     int number2 = res2;
152     int temp = 0;
153     int ggt = 0;
154     while (number1 % number2 != 0)
155     {
156         temp = number1 % number2;

```

```

157         number1 = number2;
158         number2 = temp;
159     }
160     ggt = number2;
161     return ggt;
162 }
163 }
```

### A.2.5 LogManager.cs

```

7 public class LogManager
8 {
9     private Text logText;
10    private Text aspectInfo;
11    private Text infoText;
12
13    public LogManager()
14    {
15        logText = GameObject.Find("LogText").GetComponent<Text>();
16        aspectInfo = GameObject.Find("aspectInfo").GetComponent<Text>();
17        infoText = GameObject.Find("infoText").GetComponent<Text>();
18    }
19
20    public void LogWrite(string entry)
21    {
22        logText.text += entry + "\n";
23    }
24
25    public void ShowAspectRatio(int aspectx, int aspecty, bool valid)
26    {
27        if (valid)
28        {
29            aspectInfo.color = Color.green;
30        }
31        else
32        {
33            aspectInfo.color = Color.red;
34        }
35        aspectInfo.text = aspectx + ":" + aspecty;
36    }
37
38    public void ShowSaveSettingsInformation(string text, bool fail)
39    {
40        if (fail)
41        {
42            infoText.color = Color.red;
43        } else
44        {
45            infoText.color = Color.green;
46        }
47        infoText.text = text;
48    }
49 }
```

### A.2.6 ProjectionShader.shader

```
1 Shader "Unlit/projectioin"
```

```

2 {
3     Properties
4     {
5         _MainTex ("Texture", 2D) = "white" {}
6     }
7     SubShader
8     {
9         Tags { "RenderType"="Opaque" }
10        LOD 100
11
12        Pass
13        {
14            CGPROGRAM
15            #pragma vertex vert
16            #pragma fragment frag
17
18            #include "UnityCG.cginc"
19
20            struct appdata
21            {
22                float4 vertex : POSITION;
23                float2 uv : TEXCOORD0;
24            };
25
26            struct v2f
27            {
28                float2 uv : TEXCOORD0;
29                float4 vertex : SV_POSITION;
30            };
31
32            sampler2D _MainTex;
33            float4 _MainTex_ST;
34
35            v2f vert (appdata v)
36            {
37                v2f o;
38                o.vertex = UnityObjectToClipPos(v.vertex);
39                o.uv = TRANSFORM_TEX(v.uv, _MainTex);
40                return o;
41            }
42
43 #define PI 3.14159265358979323f
44
45            float2 AzimuthalToEquiangular(float2 uv) {
46                float2 coord = (uv - .5) * 4;
47
48                float radius = length(coord);
49                float angle = atan2(coord.y, coord.x)+PI;
50
51                // from Azimuthal projection to Equiangular projection
52                float latitude = angle;
53                float longitude = 2 * acos(radius / 2.) - PI / 2;
54                return float2(latitude, longitude);
55            }
56
57            fixed4 frag(v2f i) : SV_Target
58            {

```

```

59     float2 coord = AzimuthalToEquirectangular(i.uv);
60     // equirectangular to mercator
61     float x = coord.x;
62     float y = log(tan(PI / 4. + coord.y / 2.));
63
64     x = x / (2*PI);
65     y = (y + PI) / (2*PI);
66
67     fixed4 col = tex2D(_MainTex, float2(x,y));
68
69     //make white background
70     col = length(i.uv * 2 - 1) > 1 ? 1 : col;
71     return col;
72 }
73 ENDCG
74 }
75 }
76 }
77

```

### A.2.7 maphelper.js

```
1 var map;
2
3 function initMap() {
4     var earthcenter = { lat: 0, lng: 0 };
5     map = new google.maps.Map(
6         document.getElementById('map'),
7         { zoom: 1, center: earthcenter }
8     );
9     let flights = getData();
10 }
11
12 function printMarkers(markers){
13     markers.forEach(
14         function (element) {
15             element.setMap(map);
16         }
17     )
18 }
```

### A.2.8 openskydata.js

```
1 function getData() {
2
3     fetch('https://opensky-network.org/api/states/all')
4         .then(
5             function (response) {
6                 if (response.status !== 200) {
7                     console.log('Looks like there was a problem. Status Code: ' +
8                         response.status);
9                     return;
10                }
11                response.json().then(function (data) {
12                    convertToFlightObject(data);
13                });
14            }
15        );
16    }
17}
```

```

14         }
15     )
16     .catch(function (err) {
17       console.log('Fetch Error :-S', err);
18     });
19   }
20
21 function convertToFlightObject(data){
22   var flightstates = data.states;
23   var flights = [];
24   flightstates.forEach(
25     function(element) {
26       let title = element[0];
27       let longitude = parseFloat(element[5]);
28       let latitude = parseFloat(element[6]);
29       let position = { lat: latitude, lng: longitude};
30
31       var marker = new google.maps.Marker({
32         position: position,
33         title:title
34       });
35       flights.push(marker)
36     });
37   printMarkers(flights);
38 }
```